CENICS 2020

The Thirteenth International Conference on Advances in Circuits, Electronics and Micro-electronics

ISBN: 978-1-61208-823-5

November 21 - 25, 2020

CENICS 2020 Editors

Sorore Benabid, ESEO, France

CENICS 2020

Forward

The Thirteenth International Conference on Advances in Circuits, Electronics and Micro-electronics (CENICS 2020), held on November 21-25, 2020, continued a series of events initiated in 2008, capturing the advances on special circuits, electronics, and micro-electronics on both theory and practice, from fabrication to applications using these special circuits and systems. The topics covered fundamentals of design and implementation, techniques for deployment in various applications, and advances in signal processing.

Innovations in special circuits, electronics and micro-electronics are the key support for a large spectrum of applications. The conference is focusing on several complementary aspects and targets the advances in each on it: signal processing and electronics for high speed processing, micro- and nano-electronics, special electronics for implantable and wearable devices, sensor related electronics focusing on low energy consumption, and special applications domains of telemedicine and ehealth, bio-systems, navigation systems, automotive systems, home-oriented electronics, bio-systems, etc. These applications led to special design and implementation techniques, reconfigurable and self-reconfigurable devices, and require particular methodologies to be integrated on already existing Internet-based communications and applications. Special care is required for particular devices intended to work directly with human body (implantable, wearable, ehealth), or in a human-close environment (telemedicine, house-oriented, navigation, automotive). The mini-size required by such devices confronted the scientists with special signal processing requirements.

We take here the opportunity to warmly thank all the members of the CENICS 2020 technical program committee, as well as all the reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and effort to contribute to CENICS 2020.

We also thank the members of the CENICS 2020 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope that CENICS 2020 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the field of circuits, electronics and micro-electronics.

CENICS 2020 Chairs

CENICS 2020 Technical Program Committee Publicity Chair
Javier Rocher, Universitat Politecnica de Valencia, Spain
Jose Luis García, Universitat Politecnica de Valencia, Spain

CENICS 2020

Committee

CENICS 2020 Technical Program Committee Publicity Chair

Javier Rocher, Universitat Politecnica de Valencia, Spain
Jose Luis García, Universitat Politecnica de Valencia, Spain

CENICS 2020 Technical Program Committee

Francesco Aggogeri, University of Brescia, Italy
Amirreza Ghadimi Avval, University of Arkansas, USA
Timm Bostelmann, FH Wedel (University of Applied Sciences), Germany
Manuel José Cabral dos Santos Reis, IEETA / University of Trás-os-Montes e Alto Douro, Portugal
Juan-Vicente Capella-Hernández, Universitat Politècnica de València, Spain
Saurabh Chaubey, Analog Devices Inc., Colorado Springs, USA
Fan Chen, Duke University, USA
David Cordeau, XLIM | University of Poitiers, France
Francisco Falcone, UPNA-ISC, Spain
Laurent Fesquet, TIMA Laboratory | Grenoble Institute of Technology, France
Patrick Girard, LIRMM, France
Wenkai Guan, Marquette University, USA
Amir M. Hajisadeghi, Amirkabir University of Technology (Tehran Polytechnic), Iran
Abdus Sami Hassan, Chosun University, Korea
Wen-Jyi Hwang, National Taiwan Normal University, Taipei, Taiwan
Md Toriqul Islam, University of Virginia, USA
Zhenge Jia, University of Pittsburgh, USA
Mouna Baklouti Kammoun, University of Sfax, Tunisia
Andrei Karatkevich, AGH University of Science and Technology, Krakow, Poland
Kenneth B. Kent, IBM Centre forAdvanced Studies - Atlantic | Universityof New Brunswick, Canada
Oliver Knodel, Helmholtz-Zentrum Dresden-Rossendorf, Germany
Ioannis Kouretas, University of Patras, Greece
Junghee Lee, Korea University, South Korea
Kevin Lee, School of InformationTechnology | Deakin University, Melbourne, Australia
Shuai (Steven) Li, Swansea University,UK
Diego Liberati, National Research Council of Italy, Italy
Yo-Sheng Lin, National Chi Nan University, Taiwan
David Lizcano, Madrid Open University (UDIMA), Spain
Sachin Maheshwari, Centre for Electronics Frontier | University of Southampton, UK
Amalia Miliou, Aristotle University of Thessaloniki, Greece
Bartolomeo Montrucchio, Politecnico di Torino, Italy
Jose Manuel Molina Lopez, Universidad Carlos III de Madrid, Spain
Rafael Morales Herrera, University of Castilla-La Mancha, Spain
Ioannis Moscholios, University of Peloponnese, Greece
Anish NK, Arizona State University, USA

Copyright Information

For your reference, this is the text governing the copyright release for material published by IARIA.

The copyright release is a transfer of publication rights, which allows IARIA and its partners to drive the dissemination of the published material. This allows IARIA to give articles increased visibility via distribution, inclusion in libraries, and arrangements for submission to indexes.

I, the undersigned, declare that the article is original, and that I represent the authors of this article in the copyright release matters. If this work has been done as work-for-hire, I have obtained all necessary clearances to execute a copyright release. I hereby irrevocably transfer exclusive copyright for this material to IARIA. I give IARIA permission or reproduce the work in any media format such as, but not limited to, print, digital, or electronic. I give IARIA permission to distribute the materials without restriction to any institutions or individuals. I give IARIA permission to submit the work for inclusion in article repositories as IARIA sees fit.

I, the undersigned, declare that to the best of my knowledge, the article is does not contain libelous or otherwise unlawful contents or invading the right of privacy or infringing on a proprietary right.

Following the copyright release, any circulated version of the article must bear the copyright notice and any header and footer information that IARIA applies to the published article.

IARIA grants royalty-free permission to the authors to disseminate the work, under the above provisions, for any academic, commercial, or industrial use. IARIA grants royalty-free permission to any individuals or institutions to make the article available electronically, online, or in print.

IARIA acknowledges that rights to any algorithm, process, procedure, apparatus, or articles of manufacture remain with the authors and their employers.

I, the undersigned, understand that IARIA will not be liable, in contract, tort (including, without limitation, negligence), pre-contract or other representations (other than fraudulent misrepresentations) or otherwise in connection with the publication of my work.

Exception to the above is made for work-for-hire performed while employed by the government. In that case, copyright to the material remains with the said government. The rightful owners (authors and government entity) grant unlimited and unrestricted permission to IARIA, IARIA's contractors, and IARIA's partners to further distribute the work.

Table of Contents

Predicting Noise Power in Gm-C Filters through Machine Learning

Malinka Ivanova
Technical University of Sofia
College of Energy and Electronics
Sofia, Bulgaria
E-mail: m_ivanova@tu-sofia.bg

Abstract—**Noise level in Gm-C filters is connected to the dynamic range and to the ratio signal-noise. Noise depends on the design of the transconductor cell and filter topology. Predicting the noise power before filter realization could save engineers efforts and resources. The aim of the paper is to present a novel approach for predicting the total noise power in biquad low pass second order Gm-C filter through application of machine learning algorithms as data is taken from synthesized filter topology and filter mathematical model. Five machine learning algorithms: Artificial Neural Networks, Decision Tree, Random Forest, Gradient Boosted Trees, Support Vector Machine are applied for data training and they are evaluated in order to find the most suitable for this problem. The results show that the best solutions for solving this engineering task are Artificial Neuron Networks and Decision Tree algorithms, which are characterized with the best performance and high accuracy.**

Keywords - machine learning; Gm-C filter; noise power; predictive model; signal flow graph

I. INTRODUCTION

The increased interest to the continuous-time Gm-C filters is connected to their features like high bandwidth, possibilities for parameters tuning in large frequency diapason and very low passive sensitivity, as well as their successful applications in high frequency computer, communication and bio-medical devices and systems [1]-[3].

The most discussed Gm-C filters are realized through CMOS technology as the main building block is the transconductor (OTA – operational transconductance amplifier), which is implemented in the form of differential amplifier, cascodes or folded cascode [4]. Thus, the filter properties in significant way depend on the OTA design.

The minimal input signal is limited by the input referred noise and the maximum input signal is connected to the transconductor nonlinearity. The output dynamic range is related to the total output noise and the maximal value of output linear voltage swing. So, the topics about noise reduction and transconductor linearity are still under extensive investigation.

Noise depends on the design of the transconductor cell and on the Gm-C filters topology. The research efforts are focused on minimization the noise level in the filters that will lead to the larger dynamic range and higher ratio signal/noise. The dominant noise in Gm-C filters is thermal noise, but flicker noise is also taken into consideration. The sources of noise are MOS transistors as thermal noise (white noise) is generated in the channel as consequence of random charge carriers movement and flicker noise (or $1/f$, or pink noise) is product of random mobile carriers trapping and detrapping in the channel and in the gate oxide.

Another question under exploration is related to the noise modeling and analysis and several methods are known for description the noise features and Gm-C filters noise behavior. All of them are based on noise analysis for a concrete filter solution. Exception is the general method proposed in [5][6]. The authors have developed a general structure of Gm-C filter that is a base for deriving any particular topology and analytical description. Such approach is suitable for implementation in the form of CAD tools. Nowadays, the most utilized methodology for Gm-C filters noise analysis (that is used in this work) consists of four steps: (1) identification the noise sources and noise spectrum Sn; (2) discovering the transfer function H from the noise source to the filter output; (3) calculating the output noise spectrum taking into account all noise sources; (4) calculating the total noise power as an integral over the frequency band of noise spectrum.

One contemporary approach for modeling and analysis of electronic circuits and their parameters relays on algorithms in the areas of artificial intelligence, machine learning and deep learning [7][8], which scope is presented on Figure 1.

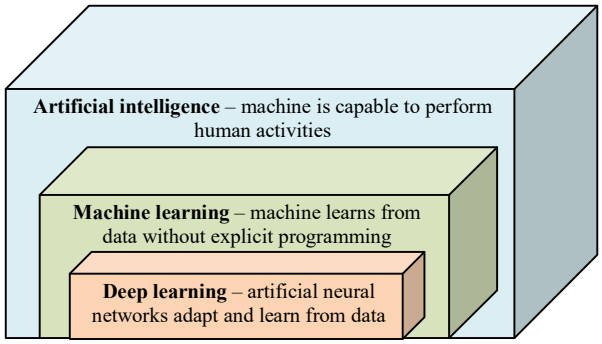


Figure 1. Scope of artificial intelligence, machine learning and deep learning

Artificial intelligence utilizes programs to reproduce human behavior and typical human activities. Machine learning is an application of artificial intelligence where machines are capable to learn from data without explicit programming. Machine learning algorithms are used for

 ISBN: 978-1-61208-823-5

solving classification and clustering tasks with aim some items, events and processes to be predicted and analyzed [9]. Deep learning is part of machine learning and it is based on Artificial Neural Networks usage that is inspired by brain functionality of biological systems. Among the advantages of algorithms for deep learning are learning from data in an easy way, correct features selection and pattern recognition. Recently, machine learning approach is applied in support of electronics engineers, facilitating and automating tasks related to computer-aided design and analysis of electronic circuits [10].

The aim of the paper is to present a novel approach for predicting noise power in Gm-C filter through application of machine learning algorithms as data is taken from synthesized filter topology and filter mathematical model. Five machine learning algorithms: Artificial Neural Networks (ANN), Decision Tree (DT), Random Forest (RF), Gradient Boosted Trees (GBT), Support Vector Machine (SVM) are applied for data training and they are evaluated in order to find the most suitable for this problem, e. g., those with the best performance and high accuracy. The rest of the paper is organized as follows: 2nd section describes the research method, the 3rd section explains the filter modeling with noise sources, the 4th section presents predictive modeling through machine learning, and the final section includes conclusion and future work.

II. RESEARCH METHOD

The proposed method for predicting the noise power in Gm-C filter is shown on Figure 2 and it consists of the following steps:

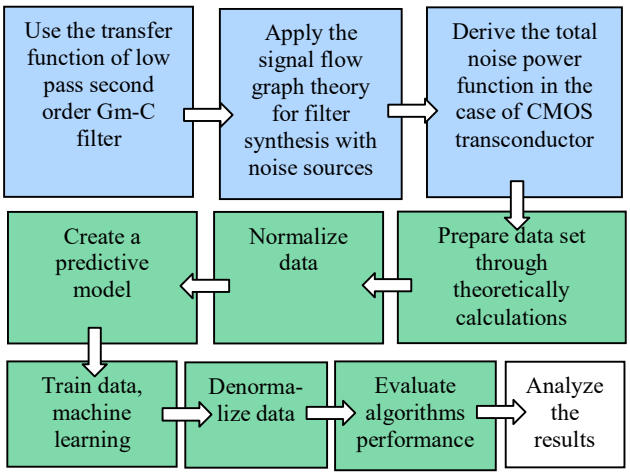


Figure 2. Used research method

(1) Derive the mathematical description of the filter noise power as before that the filter synthesis with noise sources is performed through the filter transfer function and signal flow graphs; (2) Form data set according to the filter mathematical description and data pre-processing; (3) Create a predictive model, train data, and apply machine learning algorithms; (4) Evaluate the performance of machine learning algorithms and analyze results.

III. FILTER MODELING WITH NOISE SOURCES

The literature examination shows that enough efficient noise models could be received after assumption that the capacitors in the Gm-C filter configuration are noiseless. Also, noisy OTA with transconductance g_m is modeled with a noiseless transconductor and an equivalent input referred noise voltage source U_n, which spectral density is $S_n(f)$ as it is presented on Figure 3 [11]-[13].

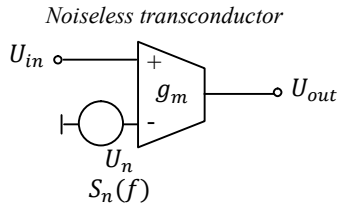


Figure 3. Noise model of an OTA [11]-[13]

In the same literature sources, it is shown that the spectral density $S_n(f)$ of one input referred noise voltage source U_n can be modeled with two components that define the influence of thermal $\frac{S_{th}}{g_m}$ and flicker $\frac{S_f}{f}$ noises:

$$S_n(f) = \frac{S_{th}}{g_m} + \frac{S_f}{f} = \frac{8kT}{3g_m} + \frac{A}{C_{ox}WLf} = K' + \frac{K''}{f}, \quad (1)$$

where $k = 1{,}38.10^{-23}J/K$ - Boltzmann constant, T - absolute temperature, A is a flicker noise coefficient that depends on the CMOS process and its value is about $10^{-25}V^2F$ (according to [14]), W and L are channel parameters of MOS transistors, C_{ox} - oxide capacitance per unit area. The total output noise voltage spectral density taking into account the Gm-C filter topology can be calculated through the following formula:

$$S_{ntotal}(f) = \bar{v}_n^2 = \sum_{i=1}^{k} S_{n_i}(f)|H_i(j2\pi f)|^2\,, \quad (2)$$

where H_i is the noise transfer function from this noise source to the filter output. The total noise power of noise spectrum is the integral over the frequency band:

$$P_{nout} = \int_0^\infty S_{ntotal}(f)df\,. \quad (3)$$

To demonstrate the noise modeling in Gm-C filters, the signal graph flow theory [15] is used. Some scientific works discuss the utilization of signal flow graph for RC and Gm-C filters design [16] [17], but here, it is applied in the context of noise power formulation, that is a new approach. The transfer function of low pass second order biquad Gm-C filter is used [18]:

$$T(s) = \frac{U_{out}}{U_{in}} = \frac{a_o}{s^2+b_1s+b_0} = \frac{\omega_o^2}{s^2+\frac{Q}{\omega_o}s+\omega_o^2}\,. \quad (4)$$

The filter synthesis is performed after several transformations of (4) and drawing the corresponding signal flow graph.

Firstly, the transfer function (4) is presented in the form $U_{out}(s^2 + b_1s + b_0) - a_oU_{in} = 0$ and obtained expression is multiplied to the variable $\frac{1}{s^2}$. The received formula

$$U_{out} = \frac{a_0}{s^2}U_{in} - \frac{b_1}{s}U_{out} - \frac{b_0}{s^2}U_{out} \quad (5)$$

 ISBN: 978-1-61208-823-5

is a base for signal graph construction (Figure 4a and b).

Further, equivalent transformation of the flow graph from Figure 4 is presented on Figure 5, where $a_o = b_o = \omega_o^2$ and $b_1 = \frac{Q}{\omega_o}$ (taking into account (4)).

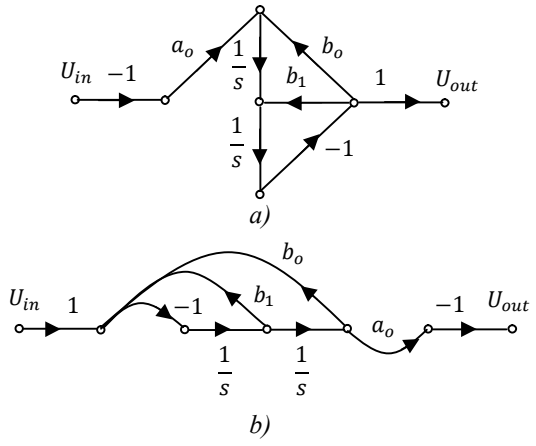


Figure 4. Signal Flow Graph of second order buquad filter: a) direct representation of (5); b) equivalent transformation

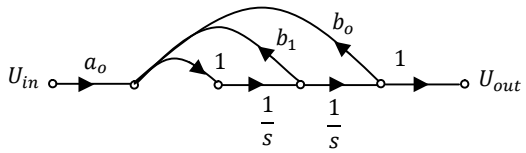


Figure 5. Equivalently transformed signal graph

The circuit implementation of this graph could be realized in different ways. One approach is shown on Figure 6, which includes two lossy integrators and a current injection source. Such topology for first time is reported in [19]. The method of current injection node is chosen because of its design flexibility and possibility for construction filters with different complexity. The first lossy integrator consists of g_{m_2}, g_{m_5}, g_{m_6}, C_1 and characterizes with the transfer function

$$H_1(s) = \frac{\frac{g_{m_2}}{C_1}}{s+\frac{g_{m_2}g_{m_5}}{C_1\ g_{m_6}}} = \frac{g_{m_2}g_{m_6}}{sC_1g_{m_6}+g_{m_2}g_{m_5}} \quad (6)$$

and noise voltage spectral density

$$S_{out1}(f) = \frac{g_{m_2}^2 S_{n_2}(f)+g_{m_5}^2 S_{n_5}(f)+g_{m_6}^2 S_{n_6}(f)}{(2\pi f C_1)^2 g_{m_6}^2+g_{m_2}^2 g_{m_5}^2}. \quad (7)$$

The second lossy integrator is implemented with g_{m_3}, g_{m_4}, C_2 and has the transfer function

$$H_2(s) = \frac{\frac{g_{m_3}}{C_2}}{s+\frac{g_{m_3}g_{m_4}}{C_2\ g_{m_3}}} = \frac{g_{m_3}}{sC_2+g_{m_4}} \quad (8)$$

and noise spectral density

$$S_{out2}(f) = \frac{g_{m_3}^2 S_{n_3}(f)+g_{m_4}^2 S_{n_4}(f)}{g_{m_4}^2+(2\pi f C_2)^2}$$ [20]. (9)

The injected current source is realized through single OTA with transconductance g_{m_1}, which transfer function is

$$H_3(s) = \frac{g_{m_1}g_{m_3}g_{m_5}}{C_1C_2g_{m_6}} \quad (10)$$

and noise spectral density

$$S_{out3}(f) = \frac{g_{m_1}^2 S_{n_1}(f)+g_{m_3}^2 S_{n_3}(f)+g_{m_5}^2 S_{n_5}(f)+g_{m_6}^2 S_{n_6}(f)}{(2\pi f C_1)^2(2\pi f C_2)^2 g_{m_6}^2}. \quad (11)$$

The total output spectral density is calculated through equation (2):

$$S_{ntotal}(f) = S_{out1}(f) + S_{out2}(f) + S_{out3}(f). \quad (12)$$

If the suggestion is that $S_{n_1}(f) = S_{n_2}(f) = S_{n_3}(f) = S_{n_4}(f) = S_{n_5}(f) = S_{n_6}(f) = S_n(f)$, $g_{m_1} = g_{m_2} = g_{m_3} = g_{m_4} = g_{m_5} = g_{m_6} = g_m$ and $C_1 = C_2 = C$, then the total output noise power is

$$P_{nout} = \int_0^\infty S_{ntotal}(f) = \int_0^\infty S_n(f)\left(\frac{3g_m^2}{(2\pi fC)^2 g_m^2+g_m^4} + \frac{2g_m^2}{(2\pi f\ )^2+g_m^2} + \frac{4}{(2\pi fC)^4}\right) \approx \int_0^\infty \frac{K_1}{f^2} + \frac{K_2}{f^3}, \quad (13)$$

where $K_1 \div K_2$ are constant values.

After simplification and integration the formula (13), for the total output noise power is received (K_1' and K_2' are constants):

$$P_{nout} = -\left(\frac{K_1'}{f} + \frac{K_2'}{2f^2}\right). \quad (14)$$

The data about the transconductor cell is taken from [18] where for simulation is chosen $0.5\mu m\ CMOS$ technology, $W/L = 10$, $\mu_n C_{ox}/2 = 5{,}78.10^{-5} A/V^2$, $g_m = 8{,}5.10^{-4} S$.

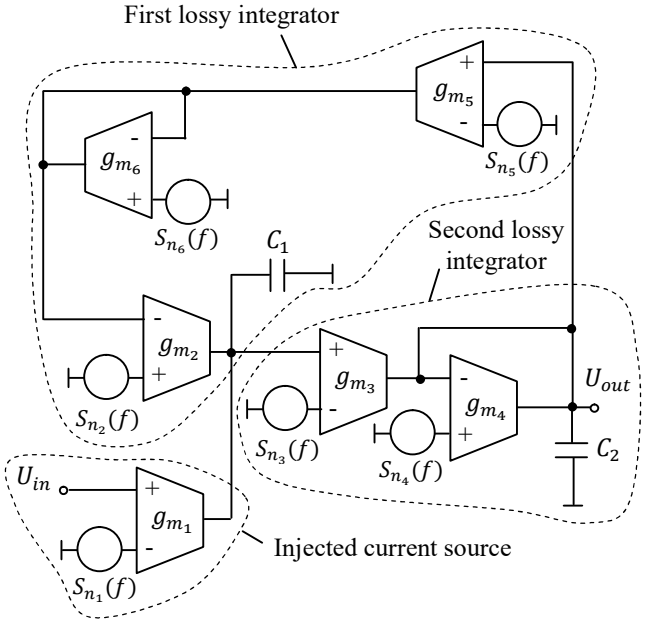


Figure 6. Gm-C filter with noise sources

Graphics of thermal and flicker noise power are presented on Figure 7. It is constructed according to obtained data of the mathematical model for noise power in

Gm-C filter. It can be seen that the flicker noise component appears at low frequency and it is much smaller (xE-25) that the thermal one (xE-17).

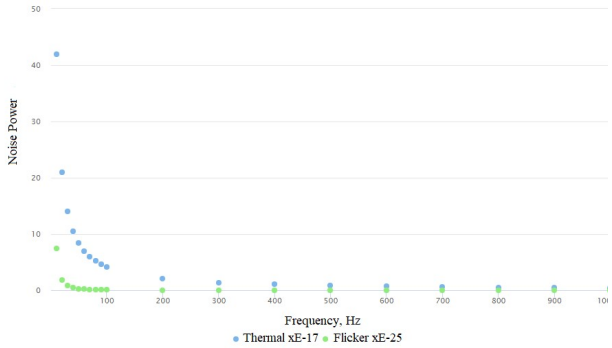


Figure 7. Thermal and flicker noise power in Gm-C filter

IV. MACHINE LEARNING AND PREDICTIVE MODELING

In this section, the development of a predictive model related to predicting the noise power in the Gm-C filter is presented. It is based on data derived theoretically from Eq. (7) and on application of supervised machine learning algorithms: ANN, DT, RF, GBT, SVM [21]-[23].

The research method is demonstrated in details for ANN algorithm, but the same method is applied to the other machine learning algorithms. The performance of the utilized machine learning algorithms is compared and discussed.

Before data training in RapidMiner Studio (version 9.4.001) environment [24], the data is normalized in the interval [0, 1], according to the standard min-max normalization: $x = \frac{x - x_{min}}{x_{max} - x_{min}}$.

Deep learning is realized through a multi-layer back-propagation neural network for which training is used stochastic gradient descent.

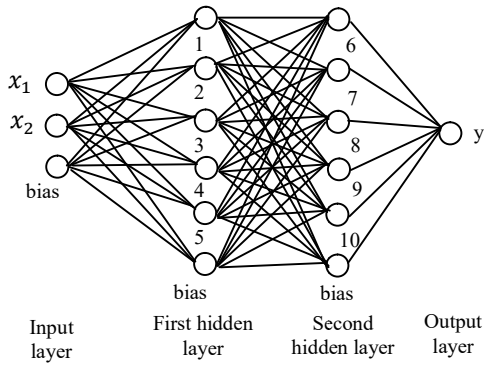


Figure 8. The constructed neural network

The artificial neural network consists of two inputs x_1 and x_2 (thermal and flicker noise power), output y (predicted noise power) and two hidden layers with five neurons in each layer (Figure 8). The neurons from the hidden layers are activated through ReLU (Rectified Linear Unit) function: $ReLU(x) = \begin{cases} 0, x \leq 0 \\ x, x > 0 \end{cases} = max\{0, x\}$.

Normalized and predicted data sets for noise power are presented on Figure 9.

Row No.	output	prediction(o...	input1	input2
1	0.372	0.403	0.495	0.250
2	0.219	0.220	0.327	0.111
3	0.152	0.146	0.242	0.062
4	0.116	0.109	0.192	0.040
5	0.077	0.070	0.134	0.020
6	0.066	0.059	0.116	0.015
7	0.057	0.051	0.102	0.012
8	0.008	0.006	0.015	0
9	0	0.001	0	0
10	0.116	0.109	0.192	0.040

Figure 9. Deep learning and predicted output

The deviation of predicted values $\hat{y}$ in comparison with the theoretically calculated y can be seen through the prediction chart on Figure 10 (the x-axis shows true values and y-axis presents predicted values).

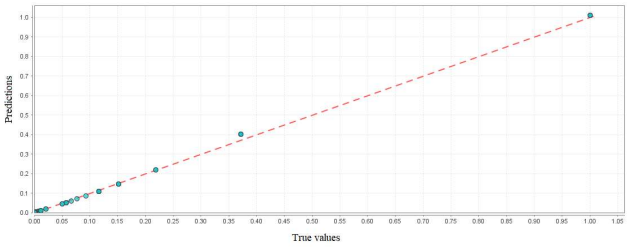


Figure 10. Deep learning – prediction chart

It can be said that the model accuracy is high, which is proved with calculation of very low errors:

- *Root Mean Square Error* (RMSE) measures the difference between N actual y and predicted $\hat{y}$ values, e. g., $RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}}$): 0.008 ± 0.005;
- *Absolute Error* (AE) is the average absolute deviation between predicted and actual value $\Delta y = \frac{|\hat{y}_i - y_i|}{N}, i = 1, 2, \dots, N$): 0.005 ± 0.002;
- *Relative Error Lenient* (REL) is the average absolute deviation between predicted and actual value divided by the maximum of predicted and actual value $REL = \frac{|\hat{y}_i - y_i|}{N} \frac{1}{\max(\hat{y}_i, y_i)}, i = 1, 2, \dots, N$): $11.85\% \pm 4.51$;
- *Squared Error* (SE) is the averaged squared error:0.000.

To construct the curves of thermal noise power taking into account theoretically calculated and predicted values, the predictions are denormalized: $x_{denorm} = x_{norm}(x_{max} - x_{min}) + x_{min}$ (Figure 11).

The prediction charts of the rest machine leaning algorithms are presented on Figure 12.

Several constructed trees through tree-based classification algorithms are shown on Figure 13 (the depicted values are normalized). The presented information through these trees could support the decision making process of designers or analysts. Following the path of the tree nodes, it is possible the decision explicitly to be explained and also the exact decision to be pointed out, concerning the values in the leafs. For example, if the tree on Figure 13b is examined and a path is followed from the root node to the leaf, it can be said that

$$IF\ 0.411 < input1 \leq 0.748\ AND\ input2 > 0.045\ THEN\ the\ predicted\ output\ IS\ 0.372.$$

The performance evaluation of the applied machine learning algorithms is shown through Table 1. The algorithm performance is important evidence about how the algorithm handles and processes the data model. The task in this work is to find the best predictive model, which is capable to solve the engineering problem related to prognosis of noise power in a Gm-C filter. Thus, an appropriate algorithm should be selected. For this purpose, the used algorithms are compared according to their accuracy (error rate), which is the most applied metrics in practice.

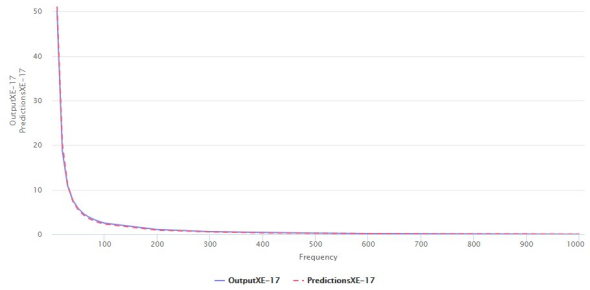
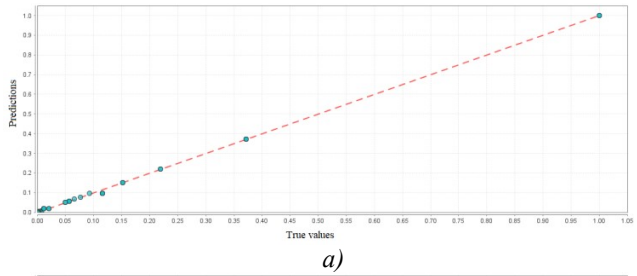
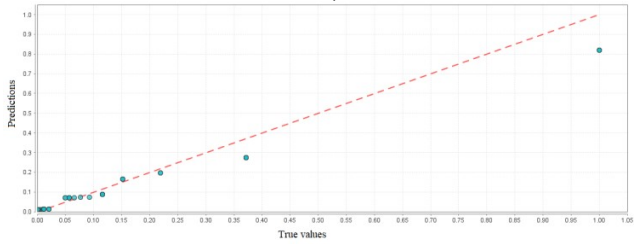


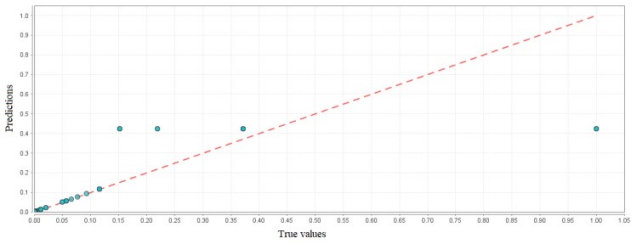
Figure 11. Theoretically calculated and predicted noise power



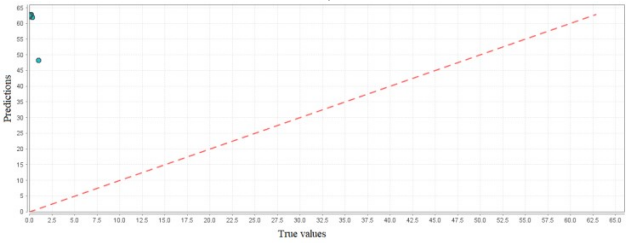
a)



b)

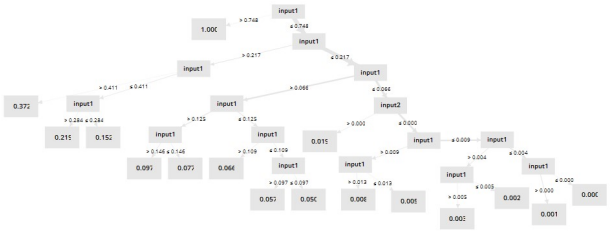


c)



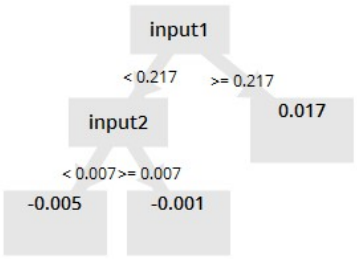
d)

Figure 12. Prediction charts of machine learning algorithms: a) Decision Tree; b) Random Forest; c) Gradient Boosted Trees; d) Support Vector Machines



a)

b)



c)

Figure 13. Constructed trees through applying: a) Decision Tree; b) Random Forest (one of 20 constructed trees); c) Gradient Boosted Trees (one of 150 constructed tress)

TABLE I. PERFORMANCE OF MACHINE LEARNING ALGORITHMS

	Criterion			
Algorithm	**RMSE**	**AE**	**REL**	**SE**
ANN	0.008 ± 0.005	0.005 ± 0.002	11.85% ± 4.51%	0.000
DT	0.007 ± 0.004	0.004 ± 0.002	6.52% ± 3.99%	0.000
RF	0.036 ± 0.029	0.025 ± 0.016	32.60% ± 10.59%	0.002 ± 0.003
GBT	0.113 ± 0.078	0.051 ± 0.033	14.94% ± 10.76%	0.018 ± 0.018
SVM	61.947 ± 0.997	61.901 ± 1.098	99.80% ± 0.13%	3838.173 ± 122.279

The comparison of prediction charts (Figure 10 and Figure 12) and data about the performance of machine learning algorithms from Table 1 point out that the ANN and DT algorithms are the best solutions for predicting the noise power in Gm-C filters. They are characterized with high accuracy.

The best performance and fastest total time (Table 2) shows the DT machine learning algorithm (the experiment is done on local computer with processor Intel(R) Core™ i7-5500U @ 2.40GHz, RAM 8GB). The worse case is the algorithm SVM that cannot deal with this predictive task. Its performance is very poor and the accuracy is small.

The comparison of the tree-based algorithms outlines that the smallest errors are introduced by DT algorithm and the biggest by GBT.

TABLE II. PROCESSING TIME

	Criterion		
Algorithm	**Training time**	**Scoring time**	**Total time**
ANN	3s	109ms	895ms
DT	61ms	65ms	251ms
RF	140ms	152ms	962ms
GBT	3s	43ms	17s
SVM	1s	65ms	4s

V. CONCLUSION

In this paper, a predictive model regarding noise power in Gm-C filter is proposed. It is created according to the designed research method. The filter synthesis with noise sources is performed through usage of its mathematical description and through applying signal flow graph theory. The derived equation for noise power allows data sets to be prepared for further statistical and machine learning processing. Five machine learning algorithms - ANN, DT, RF, GBT, and SVM are used for data training with predictive purpose. The performance of these algorithms is evaluated and they are compared according to two groups of criterion: accuracy and timing. The results show that the DT algorithm is characterized with the best performance: $RMSE = 0.007 \pm 0.004$, $AE = 0.004 \pm 0.002$, $REL = 6.52\% \pm 3.99\%$, $SE = 0.000$, $training\ time = 61ms$, $scoring\ time = 65ms$, $total\ time = 251m$. Another suitable algorithm is ANN, which is capable to predict the noise power values with very high accuracy.

It can be said that machine learning that is described as a field of artificial intelligence proposes powerful techniques and algorithms for electronic circuits' analysis and design. Studying the circuits' behavior through data about them allows a wide variety of predictive and analytical models to be created in support of engineers for decision making and problems solving. Also, machine learning gives huge opportunities for automation of engineering tasks decreasing the needed time, efforts and resources. Such approach could be implemented in CAD and EDA software in order to present a technique for design and analysis of electronic circuits and devices that could decide engineering problems with high quality and efficiency.

Learning through big data is a method that leads to better understanding the functionality and topology of electronic circuits and particularly the analog filters. Some machine learning algorithms like tree-based ones not only point out the final solution, but also describe one or several paths for its achievement. The explanation of a given solution is valuable knowledge in engineering practice. Other algorithms for deep learning which are based on artificial neural networks allow flexible and accurate approach for resolving the complexity of the problems. It seems that some machine learning algorithms are suitable for performing a given engineering task while the others cannot deal with it.

This work explores the capabilities of machine learning to predict the noise power of Gm-C filters and it is proved that the learning algorithm should be precisely chosen for obtaining the best results. Also, it is proved that a predictive model with high accuracy can be created to facilitate the performance of prognostic and analytical engineering tasks.

The lessons learned and challenges can be summarized as follows:

- At the stage of data gathering – suitable step for data collection should be chosen.
- At data model preparation – the designer should assess the data value, choosing the correct data set, ignoring the redundant data.
- At pre-processing stage, suitable format for data set processing should be selected.
- At data processing stage, several machine learning algorithms should be applied and compared for receiving the required output. The algorithms parameters should be precisely defined, because it reflects on the accuracy at the task solving.
- The predictive model should be evaluated and improved, when the obtained results are not satisfactory.

The future work will be focused on further exploration the capability of machine learning algorithms to facilitate engineering tasks, proposing possibilities for better understanding the behavior of electronic circuits. The development of predictive and analytical models will be performed, exploring their valuable meaning in support of Gm-C filters design – how the filter building blocks and elements to be chosen and arranged to form operable topology, as well as in assistance of filter analysis – what will be the filter and its building blocks reaction at different input stimuli.

 ISBN: 978-1-61208-823-5

REFERENCES

[1] M. A. Bhuiyan, M. B. Omar, M. M. Reaz, K. N. Minhad, and F. H. Hashim, "A review on CMOS Gm-C band pass filters in RF application," Journal of Theoretical and Applied Information Technology, 61(1), pp. 17-23, 2014.

[2] B. Saidulu, A. Manoharan, and K. Sundaram, "Low Noise Low Power CMOS Telescopic-OTA for Bio-Medical Applications," Computers, 5(4), 2016, https://doi.org/10.3390/computers5040025.

[3] S. K. Arfin, S. Mandal, and R. Sarpeshkar, "Dynamic-Range Analysis and Maximization of Micropower Gm–C Bandpass Filters by Adaptive Biasing", IEEE International Symposium on Circuits and Systems, pp. 2954-2957, 2009.

[4] M. Akbari and O. Hashemipour, "Design and analysis of folded cascode OTAs using Gm/Id methodology based on flicker noise reduction," Analog Integrated Circuits Signal Processing, 83, pp. 343–352, 2015.

[5] S. Koziel, S. Szczepanski and R. Schaumann, "A general approach to continuous-time Gm-C filters," International Journal of Ciruit Theory and Applications, pp. 361-383, 2003.

[6] S. Koziel, R. Schaumann and Haiqiao Xiao, "Analysis and optimization of noise in continuous-time OTA-C filters," IEEE Transactions on Circuits and Systems I, 52(6), pp. 1086–1094, 2005, doi:10.1109/TCSI.2005.849142.

[7] E. Rosenbaum, "Machine Learning: An enabling technology for electronics modeling and design optimization," 2017. [Online]. Available from: http://publish.illinois.edu/advancedelectronics/files/2017/02/DesignCon_rosenbaum_keynote-Feb-1-2017-web-.pdf

[8] T. L. Hemminger, "A Neural Network Approach to Transistor Circuit Design," Global Journal of Computer Science and Technology, 16(1), 2016, Online ISSN: 0975-4172, Print ISSN: 0975-4350.

[9] L. Rokach, A. Feldman, M. Kalech and G. Provan, "Machine-learning-based circuit synthesis," IEEE 27th Convention of Electrical and Electronics Engineers in Israel, Eilat, pp. 1-5, 2012, doi: 10.1109/EEEI.2012.6377134.

[10] Machine Learning in VLSI Computer-Aided Design, Eds. Ibrahim (Abe) M. Elfadel, Duane S. Boning, Xin Li, Springer International Publishing, 2019, doi: 10.1007/978-3-030-04666-8.

[11] T. Deliyannis, Y. Sun, and J. K. Fidler, "Two Integrator Loop OTA-C Filters," Continuous-Time Active Filter Design, Boca Raton: CRC Press LLC, 1998, https://doi.org/10.1201/9781439821879.

[12] T. Deliyannis, Y. Sun, and J. K. Fidler, "Single Operational Transconductance Amplifier (OTA) Filters", Continuous-Time Active Filter Design Boca Raton: CRC Press LLC, 1998, https://doi.org/10.1201/9781439821879.

[13] A. Brambilla, G. Espinosa, F. Montecchi and E. Sanchez-Sinecio, "Noise optimization in operational transconductance amplifier filters," IEEE International Symposium on Circuits and Systems, Portland, OR, vol.1, pp. 118-121, 1989, doi: 10.1109/ISCAS.1989.100305.

[14] Telehealth and Mobile Health, Eds Halit Eren and John G. Webster, p. 627, CRC Press, 2015, ISBN-13: 978-1482236613.

[15] G. S. Moschytz, "Analog Circuit Theory and Filter Design in the Digital World," chapter "An Introduction to Signal-Flow Graph Theory," Springer Nature Switzerland AG, 2019, pp. 85-100, doi: 10.1007/978-3-030-00096-7.

[16] P.-M. Lin, "Signal Flow Graphs in Filter Analysis and Synthesis," Circuit Analysis and Feedback Amplifier Theory, W.-K. Chen (Ed.), 2006, https://doi.org/10.1201/9781315220567.

[17] L. A. Sánchez-Gaspariano et al., "CMOS Analog Filter Design for Very High Frequency Applications," Electronics 2020, 9, 362, doi:10.3390/electronics9020362.

[18] K. Kaewdanga and W. Surakampontorn, "A balanced output CMOS OTA with wide linear current tunable range," International Journal of Electronics and Communications (AEÜ) 65, pp. 728–733, 2011.

[19] G. Espinosa, F. Montecchi, E. Sanchez-Sinencio and F. Maloberti, "Noise performances of OTA-C filters," IEEE International Symposium on Circuits and Systems 3, Espoo, Finland, pp. 2173-2176, 1988, doi: 10.1109/ISCAS.1988.15374.

[20] E. Sanchez-Sinencio, R. L. Geiger and H. Nevarez-Lozano, "Generation of continuous-time two integrator loop OTA filter structures," IEEE Transactions on Circuits and Systems, 35(8), pp. 936-946, 1988, doi: 10.1109/31.1840.

[21] M. A. Stošović and V. Litovski, "Applications of Artificial Neural Networks in Electronics," Electronics, 21(2), pp. 87-94, 2017.

[22] T. Wu, H. Zhao, F. Liu, J. Guo, and H. Wang, "Machine Learning Approach for Device-Circuit Co-Optimization of Stochastic-Memristive-Device-Based Boltzmann Machine," 2019. [Online]. Available from: https://arxiv.org/ftp/arxiv/papers/1905/1905.04431.pdf.

[23] M. Kubat, "An Introduction to Machine Learning," Springer International Publishing AG, 2017, doi: 10.1007/978-3-319-63913-0.

[24] RapidMiner, RapidMiner Studio Manual, 2014. [Online]. Available from: https://docs.rapidminer.com/downloads/RapidMiner-v6-user-manual.pdf.

 ISBN: 978-1-61208-823-5

FPGA-Based Obstacle Avoidance and Line Tracking System For Autonomous Mobile Robots

1st Sorore Benabid
Electronics and Automatic Department
Electrical and Computer Engineering School (ESEO)
Paris, France
sorore.benabid@eseo.fr

2nd Loic Latour
Aeronautical and Aerospace Systems Department
Aeronautical Engineering School (IPSA)
Paris, France
loic.latour@ipsa.fr

3rd Solène Poulain
Aeronautical and Aerospace Systems Department
Aeronautical Engineering School (IPSA)
Paris, France
solene.poulain@ipsa.fr

***Abstract*—In this paper, we develop a prototype of an autonomous mobile robot using an Field Programmable Gate Array (FPGA) based control system. The robot is able to detect the obstacles that are on its way and avoid them in real-time using ultrasonic sensors. The robot can also follow a line chosen by the operator using a camera. In addition, for the robot motion two wheels and DC motors are used. The motors are driven by an H bridge and the motor speed is controlled using a Pulse Width Modulation (PWM) signal. The whole system is implemented only in Very High speed integrated circuit Hardware Description Language (VHDL) code on a Nexys 4 development board with Artix FPGA device from Xilinx that operates at 100 MHz.**

***Index Terms*—Obstacle avoidance; Line tracking; Field Programmable Gate Array (FPGA); Autonomous mobile robots.**

I. Introduction

Mobile robots are expected to perform increasingly complex tasks in various application fields, such as: space exploration, underwater research, intelligent transport systems, military, medicine, service robots, but also in all levels of education. They should be able to navigate successfully in their environment with certain level of intelligence. To reach the desired goals, mobile robots use different kinds of sensors to collect environmental information and a set of actuators for their motion and reaction. Therefore, it is necessary to use a powerful and flexible device to control and manage the set of sensors and actuators present on the mobile robot. The Field Programmable Gate Arrays (FPGA) are gaining popularity due to their reconfigurability and parallel ability [1]. The FPGAs help enable the mobile robots to achieve their target mission successfully while guarantee real-time response in a dynamic environment [2]. Moreover, the FPGAs structure is able to execute calculating algorithms with high speed and low energy cost [3] [4].

A great deal of research has focused in the last decade on the FPGA-based mobile robot navigation. Reference [5] developed a fuzzy algorithm on FPGA-based mobile robot for line tracking and obstacle avoidance purposes. Another group proposed an FPGA-based architecture for multi-robot tracking using multiple GigE Vision cameras [6]. This architecture was implemented comprising a multi-camera frame grabber and IP cores for image processing. Another team proposed a reconfigurable embedded FPGA-based vision system for Advanced Driver Assistance Systems (ADAS) applications [7]. The developed board contains a System on Chip (SOC) composed of a programmable logic that supports parallel processing, and a microprocessor suited for serial decision making. Contrary to these various achievements, in this work, we want to realize an educational platform of a simple embedded system. This platform will allow engineering students to learn how to deploy Artificial Intelligence (AI) algorithms on an FPGA board using only the VHDL code.

In this paper, we propose an autonomous robot platform for education purposes and industrial researches. This platform completes the previous work of our team [8], in which we realized an FPGA-based vision system for autonomous mobile robots. The previous system evaluated in real-time the distance between a robot and an object or obstacle in front of it. In this work, we perform other tasks namely: autonomous navigation, obstacle avoidance and line tracking. The proposed system is implemented only in VHDL code. The paper is divided into four Sections. Section II describes the architecture of the proposed system, including the robot motion, obstacle avoidance principle and line tracking algorithm. The experiments conducted to demonstrate the performances of the system are given in Section III. Finally, we conclude in Section IV.

II. Robot Architecture

The proposed embedded system for a mobile robot is shown in Figure 1. We have used two wheels and DC motors with an H bridge controller for the robot motion. The desired direction and speed of the robot is determined by the control and processing unit. This unit is an FPGA development board that allows our robot to react in real-time to its exterior environment, depending on the input information given by a single camera and ultrasonic sensors. The ultrasonic sensors are used to allow the robot to avoid obstacles and the camera is used to ensure the line tracking. Obviously, we use a battery as an external power supply for the DC motors and also for the FPGA board. Furthermore, the camera requires a huge capacity of memory for the image processing without any quality losses. Thus, we chose the Nexys 4 board because

 ISBN: 978-1-61208-823-5

TABLE I
ROBOT SPECIFICATIONS

FPGA Board	*Nexys 4 DDR - Artix-7*
Board Dimension	10,9 x 12,2 cm
Logic Slices:	15 850
Block RAM	4,860 Kbits
DDR2 Memory	$128MiB$
Operating Frequency	$100MHz$
Power Supply	4.5V-5.5V
Camera	*CMOS Image sensor*
Photosensitive Matrix	640 x 480
Issuance formats (8)	RGB 565
Maximum rate:	30 fps in VGA
Pixel height	$3.6\mu m$
Output format	VGA
DC motor	*Decelerate motors ratio 1:48*
Drive voltage	12V
Ultrasonic sensor	*HC-SR04*
Operating voltage	$5\ V$
Operating current	$15\ mA$
Range of distance	$2\ cm$ to $400\ cm$
Battery	EC Technology
Capacity	$22400mAh$ / $82.8Wh$
Entry	$5V$ / $2A$
Max Output	$5V$ / $3.4A$ (AUTO)
Dimension	$160x80x23mm$
Weight	$462g$

of its storage capacity and the number of available ports. The main specifications of the embedded system are listed in Table I.

We describe in the next subsections the main operations performed by the robot: robot motion, obstacle avoidance and line tracking.

A. Robot Motion

To perform properly the robot motion, the DC motor needs to receive an electrical current which intensity will directly impact the speed and rotation of the motor. We used the Pulse Width Modulation (PWM) principle to control the speed of the robot. This will make use of the FPGA board frequency to periodically supply an amount of energy to the motor and thus provide efficient speed control.

B. Obstacle Avoidance

This function was filled by three ultrasonic sensors placed at the front of the robot and oriented at different angles. Ultrasonic Sensors measure the distance (D) to the target object by measuring the time (T) between the transmitted and received wave.

$$D = \frac{T.S}{2} \quad (1)$$

where S represents the sound speed.

The avoidance obstacle can be achieved through the distance measured by the different ultrasonic sensors as described by the flowchart given in Figure 2. Once a distance is inferior to a threshold that we have set by trials, we consider there is an obstacle to avoid. Then, we compare the different distances given by the three ultrasonic sensors, if the distance to the left is greater than the right one, the robot should turn left to avoid the object on the right. Moreover, if the distance measured is inferior to the threshold of every ultrasonic sensor, the robot will do a half turn to avoid getting stuck.

C. Line Tracking

In the proposed system, as shown in Figure 3, we have chosen to track a white line on a black background reversed when filtering. Then, we chose 11 pixels aligned horizontally, from the center to both sides of the image. These 11 points are spaced by 44 pixels and will be represented by an 11-bit binary vector.

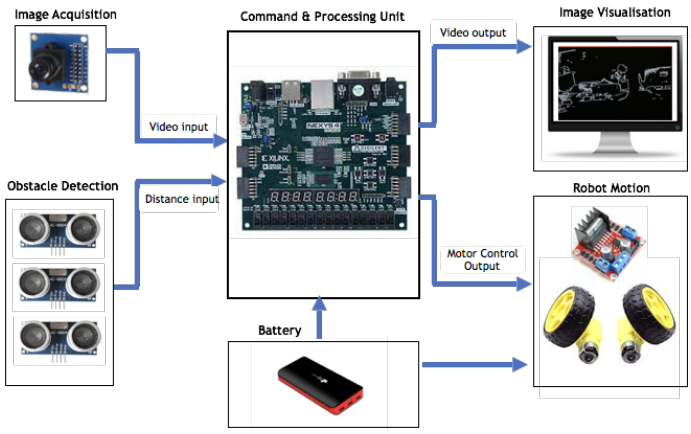


Fig. 1. Embedded system architecture for the mobile robot

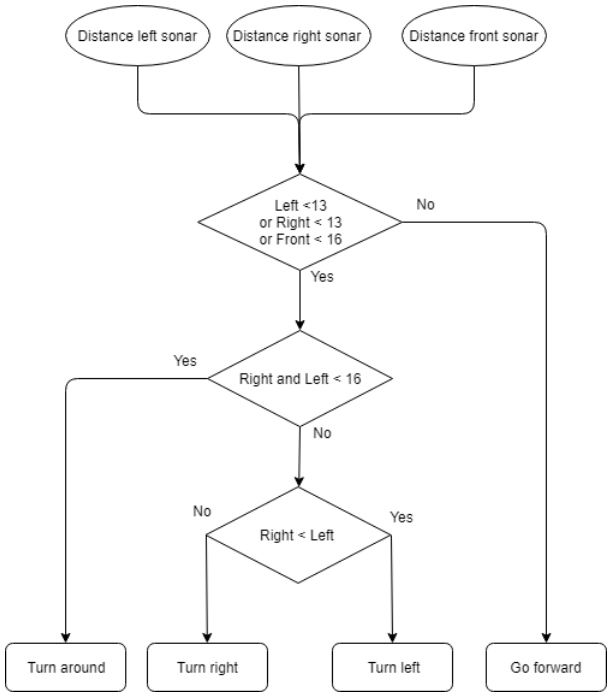


Fig. 2. Obstacle avoidance algorithm using the ultrasonic sensors

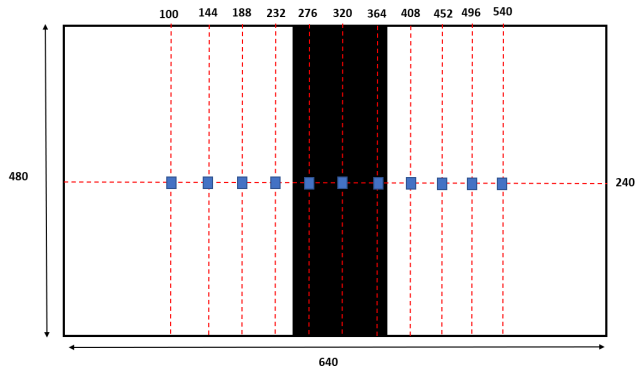


Fig. 3. Line modeling after black and white filter

To track the line, we used a single camera at the front of the robot. A black and white filter is applied to the captured image by the camera. This filter help enable to distinguish the line from the rest of the image. Each bit of the 11-bits vector are set to ”1” (white) if the pixel exceeds a threshold. Otherwise, the bits are set to ”0” (black). We adjusted the filter threshold using tests for a better result.

After the black and white filtering, the line is modeled by 3 bits at the center of the 11-bits vector that represent the 11 horizontally pixels as illustrated in Figure 3. Then, the robot can detect the line according to the value of the 3-bits modeling this line. If these 3-bits are equal to ”0” the line is detected, otherwise the white color is dominant and the line is not detected.

TABLE II
THE MOTOR CONTROL USING THE LINE MODEL

Binary vector	*Action on motor*	*Duty cycle (%)*
00111111111	Turn left	65
00011111111	Turn left	60
10001111111	Turn left	60
11000111111	Turn left	50
11100011111	Forward	50
11110001111	Forward	50
11111000111	Forward	50
11111100011	Turn right	50
11111110001	Turn right	60
11111111000	Turn right	60
11111111100	Turn right	65

Finally, we have to control the speed and the direction of the robot (left, right or center) according to the position of the ”0” in the 11-bits vector as depicted in Table II. The PWM modulation adjusts the robot speed as mentioned previously (Section II-A). However, if the robot is in a bend it loses the line, then it will stop.

III. Experimental Results

In this Section, we describe the structure of the robot, the synthesised architecture in VHDL code and the experimental results of the achieved prototype.

A. Robot Structure Design

The structure is designed on CATIA software (Computer-Aided Three-dimensional Interactive Application). It is a multi-platform software suite for Computer-Aided Design (CAD), Computer-Aided Manufacturing (CAM), Computer-Aided Engineering (CAE), developed by the french company Dassault Systèmes [9]. The designed structure is printed by a 3D printer as shown in Figure 4. This structure allowed us to realize the robot’s prototype as shown in Figure 5.

B. FPGA Architecture

The synthesised architecture is illustrated in Figure 6 and described in VHDL code. Signals from the 3 ultrasonic sensors (echo signals) are transmitted to the ”Sonar driver” block that calculated the distances of the obstacles detected by the robot. These distances are displayed by the ”7-segments” block, and are also used to determine the motion of the robot by the ”Motor driver” module. This module is used to control the

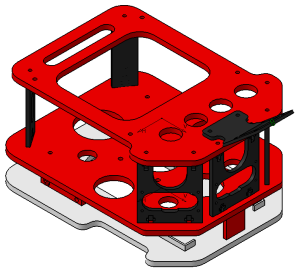


Fig. 4. Structure of the robot

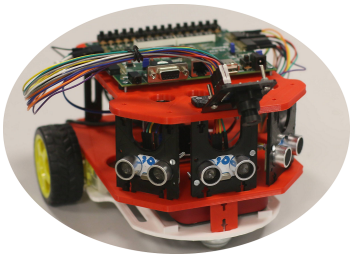


Fig. 5. The Robot

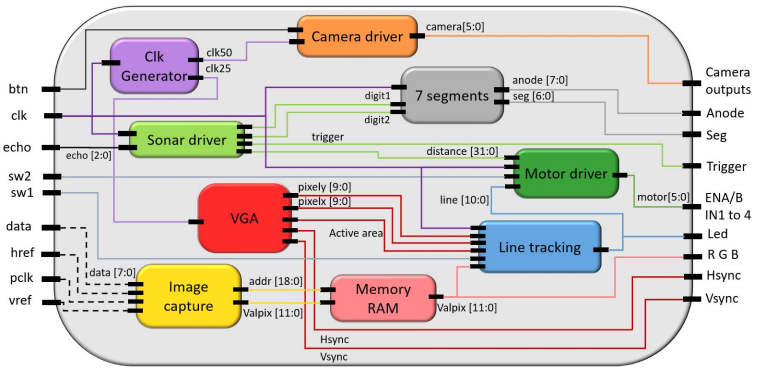


Fig. 6. FPGA-based embedded system architecture: robot motion, image processing, obstacle avoidance and line tracking

TABLE III
THE MOTOR CONTROL

IN1,IN3	*IN2,IN4*	*Result*	*ENA,ENB*
0	0	Stop	0
0	1	Rotation+	PWM
1	0	Rotation-	PWM
1	1	NOT ALLOWED	0

TABLE IV
FPGA RESOURCE USAGE OF THE PROPOSED SYSTEM

Resource	*Utilization*	*Available*	*%*
LUT	1107	63400	1,75
FF	481	126800	0,38
BRAM	104	135	77,04
IO	73	210	34,76
BUFG	5	32	15,63
MMCM	1	6	16,67

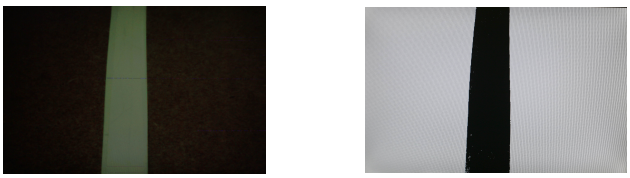


Fig. 7. Black and white filter for line tracking

DC motors via an H bridge. Using IN1, IN2, IN3 and IN4 pins we can control the direction of the rotation of the motor and thus make the robot go forward or backward (Table III). The PWM signals control the motors speed using the ENA and ENB pins. Indeed, depending on the choice of duty cycle, the intensity on these pins will be more or less important alike the speed.

The "Camera driver" block is used to configure the camera to generate the appropriate synchronization signals with clock frequency at $50MHz$. The image of the camera is captured by the "Image capture" block, then stored in the "Memory RAM" block. The image is then filtered (black and white filter) and the "Line tracking" module is used to find the position of the line as explained previously (Section II-C). The line position is then transmitted to the "Motor driver" block that allows the robot to follow the line.

Finally, the image is displayed on the VGA screen according to the synchronization signals and the pixel position generated by the "VGA" block. The "CLK generator" block provides two clock signals from the FPGA operation clock $100MHz$ (Table I); $25MHz$ for the VGA screen clock and $50MHz$ for the "Camera Driver" block.

The described architecture is synthesized and implemented in Artix-7 FPGA available on the Nexys 4 development board from Xilinx using the Vivado 2018.2 software. The proposed system does not require a processor and external RAM resources. The resource usage is summarized in Table IV.

The black and white filter for line tracking is illustrated in Figure 7. A video showing the obstacle avoidance of the proposed system can be found at: https://youtu.be/tjWPFtim8CQ. For the line tracking test, a video is available on this link: https://youtu.be/SjUswlInYgM.

IV. Conclusion

The emergence of FPGAs has given improvement of a real-time mobile robot navigation systems. We have proposed in this paper an FPGA-based embedded system navigation for the mobile autonomous robots. The proposed system can help the robots to navigate successfully by analysing the visual features of the surrounding environment. This system can detect and avoid the obstacles and can also track a line. This system uses a single front camera and ultrasonic sensors as sources of information on the exterior environment. It uses also two wheels and the DC motors to ensure the robot motion. The whole architecture is implemented only in VHDL parallel programming code and will be further improved and extended for other tasks such as path planning and object recognition algorithms. The achieved robot provides a good experimental platform for academic environment in collaboration with industrial partners. The engineering students will learn and test AI algorithms for mobile robots. They will also be able to propose a new AI techniques for several applications: Intelligent Transportation Systems (ITS), medical application, environmental protection, etc.

Acknowledgment

We would like to thank the laboratory of the autonomous aerial systems at IPSA for their daily help especially for the use of the 3D printer, different components and software tools that we have used to design and realise our prototype.

References

[1] R. Dubey, N. Pradhan, K. M. Krishna, and S. R. Chowdhury, "Field programmable gate array (FPGA) based collision avoidance using acceleration velocity obstacles," in 2012 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dec 2012, pp. 2333–2338.

[2] D. Honegger, H. Oleynikova, and M. Pollefeys, "Real-time and low latency embedded computer vision hardware based on a combination of FPGA and mobile CPU," in 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sep. 2014, pp. 4930–4935.

[3] M. Samarawickrama, A. Pasqual, and R. Rodrigo, "FPGA-based compact and flexible architecture for real-time embedded vision systems," in International Conference on Industrial and Information Systems (ICIIS), Dec 2009, pp. 337–342.

[4] J. Svab, T. Krajnik, J. Faigl, and L. Preucil, "FPGA based speeded up robust features," in 2009 IEEE International Conference on Technologies for Practical Robot Applications, Nov 2009, pp. 35–41.

[5] S. Boroumand, A. Saboury, A. Ravari, M. Tale Masouleh, and A. Fakharian, "Path tracking and obstacle avoidance of a FPGA-based mobile robot MRTQ via fuzzy algorithm," in 2013 13th Iranian Conference on Fuzzy Systems (IFSC), Aug 2013, pp. 1–5.

[6] A. Irwansyah, O. W. Ibraheem, J. Hagemeyer, M. Porrmann, and U. Rückert, "FPGA-based multi-robot tracking," J. Parallel Distrib. Comput., vol. 107, 2017, pp. 146–161.

[7] G. Velez, A. Cortés, M. Nieto, I. Vélez, and O. Otaegui, "A reconfigurable embedded vision system for advanced driver assistance," Journal of Real-Time Image Processing, vol. 10, no. 4, 2015, pp. 725–739.

[8] S. Benabid, L. Latour, S. Poulain, and M. Jaafar, "FPGA-based real-time embedded vision system for autonomous mobile robots," in 2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS), 2019, pp. 1093–1096.

[9] D. Systèmes. CATIA. [Online]. Available: https://www.3ds.com/products-services/catia/

 ISBN: 978-1-61208-823-5

Improving the Gradient Descent Based FPGA-Placement Algorithm

Tobias Thiemann, Timm Bostelmann and Sergei Sawitzki

FH Wedel (University of Applied Sciences)
Wedel, Germany
Email: {inf103917,bos,saw}@fh-wedel.de

***Abstract*—In a previous paper of the authors, a gradient descent based Field-Programmable Gate Array (FPGA) placement algorithm was presented. It achieved similar results to the reference (based on simulated annealing) regarding the bounding-box quality, while being on average 3.8 times faster. However, the critical path was significantly longer. The paper concluded by pointing out several possible areas of improvement, which could lead to better quality of the placement results and/or further increases to placement speed. These different suggestions were evaluated, and the results applied to the algorithm. This paper explains the process and shows the final results of the improved algorithm. The improvements lead to the final version of the program being roughly 5.1 times as fast as the reference, while also improving the bounding box cost by 1.27 %, as well as the timing of the critical path by 16 %, when compared to the original version.**

Keywords–EDA; FPGA; placement; gradient descent.

I. Introduction

The work presented in this paper seeks to improve on the results from the previous paper "Fast FPGA-Placement Using a Gradient Descent Based Algorithm" [1], which described the base algorithm. The underlying problem of netlist placement for FPGAs can be roughly described as selecting a resource cell (a position) on the target FPGA for every node of the given netlist. Thereby, necessary constraints (e.g., not overlapping) must be considered as well as quality constraints (e.g., the length of resulting critical path).

To differentiate, which version of the algorithm is being referred to, the terms Gradient-Place Original (GPO) and Gradient-Place New (GPN) are used, where GPO refers to the final version of the program from the original paper, while GPN refers to the program, as improved in this paper. It should be noted that the usage of GPN during the explanation of the process does not refer to the final results, but to the results up to that point, including the currently discussed improvement. Only in Section VII, when the final results are presented, does GPN refer to the final version. When just the term "the program" is used it generally means that it applies equally to the original and the improved version.

The rest of this work is organized as follows. In Section II, the implementation of GPO is introduced in form of a shortened version of the implementation sections from the previous paper [1] for easy reference. From Section III to Section VI, the process of evaluating the various possible improvements for GPN is explained and intermediate results are presented. The changes made for each step are used for all following steps, so each improvement is incremental to the previous. In Section VII, the final version of GPN is benchmarked using the Microelectronics Center of North Carolina (MCNC) set of netlists [2], which were also used in the original paper. Finally, in Section VIII, the results of this work are summarized and a prospect to further work is given.

II. Background

This section will give an overview of the different steps in the implementation of GPO, which represents a shortened version of the implementation sections from the previous paper, purely for easy reference.

A. Preparation

GPO initially assigned a random starting position to each node, which also included the nodes, which represent pins, meaning that pin locations may also be somewhere in the middle of the placement grid initially. These positions were distributed over the placement area in continuous coordinates, which means they do not represent a valid placement.

The coordinates are generated using a deterministic XOR-Shift Pseudo-Random Number Generator (PRNG) [3] with a static seed value. This means that multiple runs will generate the same initial placement and hence have the same results.

B. Legalization

Since continuous coordinates are used, GPO performs a legalization step for each iteration, to assign a valid position to each node. This is necessary since node positions during training can be anywhere on the placement grid, and there might be an arbitrary number of nodes occupying the space of a single cell of the placement grid. The function of the legalization is shown schematically in Figure 1 and Figure 2, where the first shows the current position of all nodes, and the second shows the valid positions for the nodes, as determined during legalization.

The concrete approach is a simplified version of the legalization, as presented in the work of Gort and Anderson [4]. Their algorithm works in two steps:

1) Begin with single cell regions, determine all nodes that currently occupy that region, and extend the regions outwards, merging with neighboring regions, until all nodes could fit into the extended region.
2) Recursively split the resulting region(s) into halves, and assign nodes to the new halves, depending on their exact position, while making sure no split region overfills.

 ISBN: 978-1-61208-823-5

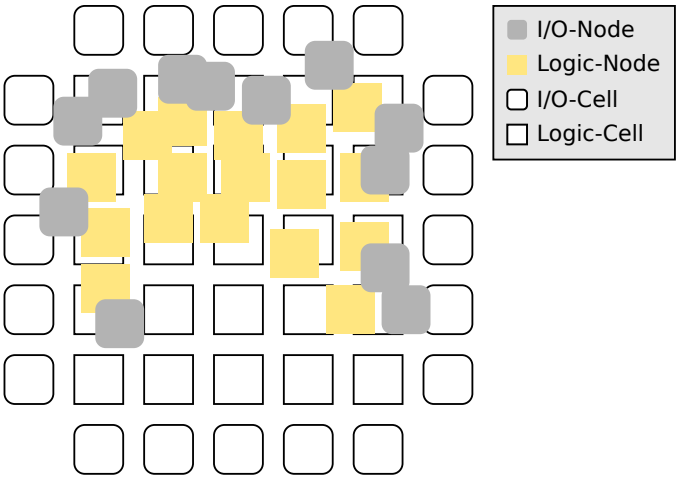


Figure 1. An exemplary placement during the optimization. The nodes are the elements of the netlist that need to be placed on the resource cells of the FPGA.

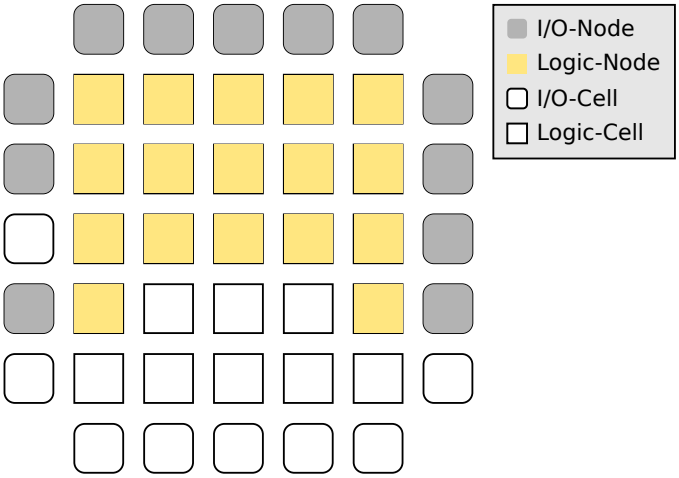


Figure 2. Legal placement determined by legalization step. The nodes are the elements of the netlist that need to be placed on the resource cells of the FPGA.

GPO uses only the second step, to simplify implementation, as well as to potentially save on time for the legalization. Concretely, the first step was dropped, since it would often result in the worst case scenario, where regions are extended until there is a single large region spanning the entire placement grid. Directly starting with that single large region instead reduces complexity, while also cutting out the time required to compute that single region.

C. Gradient Calculation

For the gradient calculation GPO utilizes a simple cost function based on the bounding boxes of the separate nets that a node is connected to. This cost function, when expressed for a specific node, is concretely:

$$C_i = \alpha_2 \cdot \sum_{n \in N_i} \Big(e^{\alpha_1 \cdot (x_i - \max_x(n))} + e^{\alpha_1 \cdot (\min_x(n) - x_i)} + e^{\alpha_1 \cdot (y_i - \max_y(n))} + e^{\alpha_1 \cdot (\min_y(n) - y_i)} \Big) \quad (1)$$

where x_i and y_i describe the x and y coordinates of the current node, N_i describes the set of all nets that the node is connected to and $\min_x$, $\max_x$, $\min_y$ and $\max_y$ are the minimal and maximal coordinates of the given net, i.e., the bounding-box. To speed up calculations the bounding boxes of the nets are determined in a separate step before the gradients are calculated.

The parameters α_1 and α_2 affect the exact shape of the cost function. α_1 determines, how large the distance between a node and the bounding-box can be before the node's effect on the cost function becomes negligible, as well as the steepness of the gradients of nodes close to the border. α_2 is a simple scaling factor, which allows increasing or decreasing the weight of the gradients relative to the legalization.

This cost function ensures that gradients will be continuous, and that nodes will have at least a small gradient for each net they are connected to (unless the node is exactly in the middle of the net), whereas a cost function using only the borders of the net as a hard threshold would cause very sporadic gradients and affect only a tiny proportion of the nodes.

The gradients for a node in x- and y-direction, based on (1), can then be calculated as:

$$\frac{dC_i}{dx_i} = \alpha_1 \alpha_2 \cdot \sum_{n \in N_i} \Big(e^{\alpha_1 \cdot (x_i - \max_x(n))} - e^{\alpha_1 \cdot (\min_x(n) - x_i)} \Big) \quad (2)$$

$$\frac{dC_i}{dy_i} = \alpha_1 \alpha_2 \cdot \sum_{n \in N_i} \Big(e^{\alpha_1 \cdot (y_i - \max_y(n))} - e^{\alpha_1 \cdot (\min_y(n) - y_i)} \Big) \quad (3)$$

The exemplary plot in Figure 3, assuming a net with the boundaries $\min_x = 1$, $\max_x = 7$ and $\alpha_2 = 1$, helps visualizing the effect of α_1 on the gradient. In general, it holds that nodes, which are near the bounding-box of their containing net, have a gradient of $\pm\alpha_1\alpha_2$, whereas the gradients of nodes with a larger distance to the bounding-box are much lower. Consequentially, nodes closer to the bounding-box will be moved more during the next optimization step.

D. Optimization

GPO uses the Adam optimization algorithm, introduced by Kingma and Ba [5], to apply the calculated gradients to the nodes. This algorithm does not just simply use the gradients as they are for each individual iteration, but forms a running average of the gradients, generally called the first moment, as well as a second moment, which is used to scale the gradients, such that changes made by the optimizer are neither excessively large, nor too small to matter. It consists of the following calculations, which are performed individually for the x- and y-axis of each node:

$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$	Running avg. 1st moment
$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$	Running avg. 2nd moment
$\hat{m}_t = m_t / (1 - \beta_1^t)$	Bias corrected 1st moment
$\hat{v}_t = v_t / (1 - \beta_2^t)$	Bias corrected 2nd moment
$\phi_t = \phi_{t-1} - S_a \cdot \hat{m}_t / \left(\sqrt{\hat{v}_t} + \epsilon \right)$	Update of the variable

The term g_t refers to the corresponding gradients, as calculated in the previous step. The constants β_1 and β_2 define how fast the moving averages of the first and second moments change. S_a refers to the step size, which allows for compromise between stability and speed of convergence. High values may approach a minimum quickly, but then fail to converge on it, whereas a too small step size results in good convergence

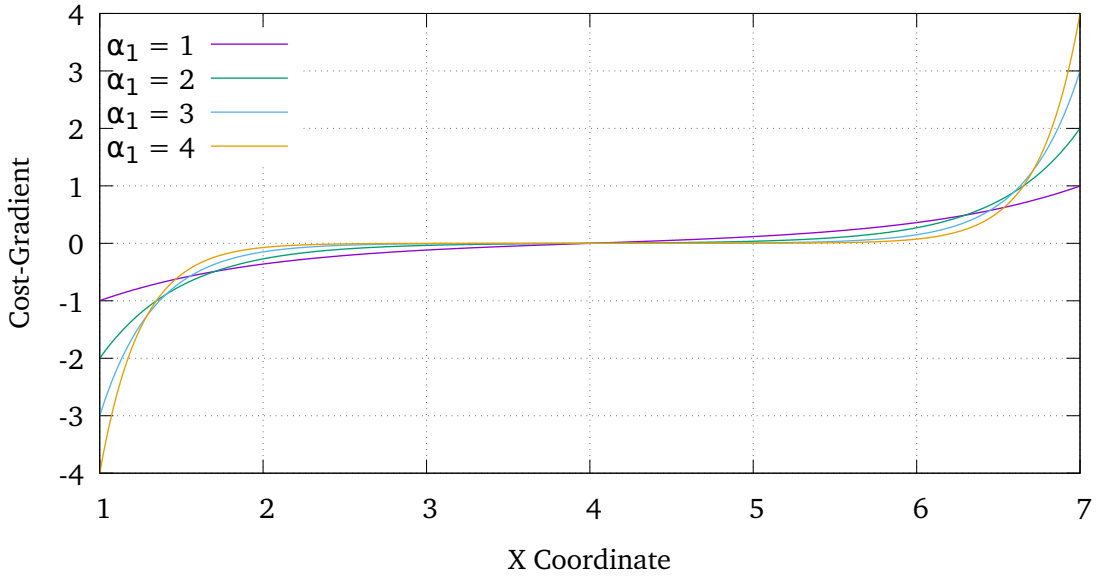


Figure 3. Exemplary plot of possible gradients for the x coordinate of a node, assuming a net with the boundaries $\min_x = 1$, $\max_x = 7$ and $\alpha_2 = 1$.

towards a minimum, but may severely increase time required for the algorithm to converge. ϵ is a small bias used to prevent divisions by zero during the variable update step.

The paper by Kingma and Ba suggests the values $S_a = 0.001$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$, which were determined by testing them on a few machine learning applications. GPO on the other hand uses slightly different values, which have been determined experimentally. These concrete values are $S_a = 1.5$, $\beta_1 = 0.96$ and $\beta_2 = 0.998$. The difference can be explained by the difference in the context they are used. Especially the step size usually needs to be much smaller in machine learning applications, since a tiny change there may already cause massive differences in the result, whereas a tiny movement of a node in the context of the placement problem has basically no effect.

The bias correction of the two moments is required during the first few iterations, where the first and second moment are still very low, which would lead to disproportionally small changes for these iterations.

E. Placement Phases

The actual placement in GPO is performed as a certain number of iterations over several phases, where each iteration performs all the mentioned steps in the same order. Each phase may slightly adjust certain parameters, which was deemed sensible to improve early arranging and later final detail placement. The different placement phases are as follows:

1) **Presorting** (5000 iterations)
   In this phase, all nodes are moved with a high step size in the general direction of their final position.
2) **Grid placement** (1000 iterations)
   In this phase, the factor for the legalization is increased. Thereby, the nodes are pulled harder towards legal positions. This is necessary – for example – to prevent input and output cells from getting stuck in the logic block section of the architecture, as well as making sure that Configurable Logic Blocks (CLBs) spread over the entire placement area.
3) **Initial detail placement** (1000 iterations)
   In this phase, the global step-factor is reduced from 1.0 to 0.1. This influences the legalization and the optimization equally, so that the balance between those two steps is not changed. However, the movements are much smaller, resulting in more localized changes.
4) **Detail placement** (5000 iterations)
   In this phase, the step-size of the optimization is reduced linearly from 1.5 to 0.3 (20 percent of its original value). This means that the relative effect of the legalization slowly becomes dominant over the optimization, forcing nodes closer to their final position and allowing only small, final node movements towards the end.
5) **Final placement** (100 iterations)
   In this phase the influence of the optimization is reduced to zero, so that effectively only the legalization is active. Hence, the nodes are moved to their final position in the grid.

III. Optimization of run time by utilizing multithreading

As previously noted, some steps of the general algorithm have no sequential dependencies and as such can be trivially executed in parallel. These are specifically:

1) Calculation of the bounding boxes and costs of the separate nets.
2) Calculation of the gradients for each node.
3) Applying the gradients for each node.

The calculation of legal positions for each node has an inherent sequential component, since it requires sorting of the list of nodes inside a given region, which prevents it from being fully parallel. However, it is still be possible to run recursive calls in parallel, since separate regions have no dependencies on each other.

The framework OpenMP [6] was used to evaluate the effect of using multiple threads for the different operations, without

TABLE I. LIST OF THE TIMINGS FOR THE SEPARATE STEPS THAT OCCURRED WHEN USING SEPARATE OPENMP TASKS FOR THE RECURSIONS OF THE LEGALIZATION FUNCTION, GIVEN A SPECIFIC THRESHOLD

Thresh.	Legal. (μs)	Calc. Grad. (μs)	Apply Grad. (μs)	Calc. Nets (μs)	Time Real (s)	Time User (s)	Time Sys (s)
-	3159.3	1272.0	63.9	268.3	57.92	57.88	0.030
2	2866.3	1263.1	63.9	259.7	54.34	128.6	47.74
4	2610.6	1244.5	64.3	268.3	51.15	118.7	39.23
8	2258.9	1231.5	64.4	250.7	46.55	105.5	27.53
16	2006.1	1229.5	64.2	246.7	43.38	97.21	18.92
32	1646.1	1251.0	64.3	262.4	39.52	84.48	7.064
64	1524.3	1228.6	65.5	253.8	37.63	78.59	3.427
128	1458.3	1218.0	64.4	240.2	36.56	75.05	1.324
256	1455.1	1219.9	64.7	243.3	36.59	74.00	1.244
512	1467.4	1226.5	64.6	248.2	36.84	73.89	1.221
1024	1503.4	1226.4	64.9	251.3	37.34	73.41	1.147
2048	1605.2	1257.5	65.7	251.1	38.91	72.91	1.302
4096	1744.5	1305.7	67.8	259.5	41.31	71.46	0.909
8192	2415.1	1300.4	67.4	250.7	49.12	70.79	0.795

requiring major changes to the program structure. OpenMP is a combination of compiler features and a run time library, which several modern compilers offer support for, and allows for easy parallel execution of certain types of loops and program constructs, without the need to explicitly include primitives for thread creation and synchronization. This is achieved by annotating loops or certain function calls with OpenMP-specific pragmas, which the compiler automatically translates into the required threading primitives. OpenMP internally employs a thread pool, with a specified number of worker threads, to avoid the overhead of frequent thread creation. It also allows to explicitly specify the number of threads to use for an operation, as well as the scheduling mode. Unless otherwise specified, OpenMP will default to the "Active" scheduling mode, which causes idle threads to busy-wait on the work queue. This potentially reduces the delay before a thread starts working on a task, but will also waste a lot of Central Processing Unit (CPU) time. The alternative is the "Passive" scheduling mode, where all idle threads are sleeping and have to be woken up once work is available.

While intuition may suggest that using more threads to accomplish a task will always cause faster processing, this is usually far from the truth in real applications. As outlined above, not all parts of the program can be run in parallel, which means that execution can never be faster than these portions. Next, thread creation incurs a certain overhead, as do the required synchronization primitives to make sure that all threads finished a specific workload. Lastly, there are often non-deterministic effects between threads, caused by a variety of factors, including the scheduling of threads performed by the Operation System (OS), as well as caching and speculative behavior employed by the CPU.

As a measure for the effect of the multithreading, various timing data were recorded. For each of the 12,000 iterations the duration for each of the steps was noted, and combined into an average at the end. Additionally, the time for the overall placement, as well as the CPU time attributed to the user and the system were noted, where the CPU time is effectively the time spent computing times the number of CPU cores that were busy during that time, and can thus exceed the real execution time. For all tests OpenMP was set up to use a maximum of eight threads, matching the number of hardware threads of the CPU used for the measurements, and the passive scheduling mode. The netlist used for the measurements was the "clma" netlist, which contains 8383 CLBs, 62 inputs and 82 outputs, and represents the largest netlist in the benchmark for the previous results, which means improvements should be most obvious on this netlist.

A. *Effect on the legalization step*

The legalization step was evaluated first, since it represents the majority of the time spent on each iteration. OpenMP is used here for parallel execution of recursive calls, as previously mentioned. This is guarded by a threshold value, depending on the number of nodes left in the region. The threshold value is used as a compromise between how many threads can work on the recursive calls, and the overhead that is incurred on task creation and thread interaction. Concretely, a low threshold value means potentially higher parallelism, while a high threshold means lower overhead.

To achieve the parallel execution, the separate recursive calls are specified as OpenMP "tasks", which are scheduled by OpenMP and will be distributed to any free threads. This means that the maximum number of threads still applies.

Table I shows the resulting timing data of the entire algorithm relative to the specified threshold value. The first row designates the baseline timing, when the entire program runs with a single thread. It can be seen that even at a threshold of two, the program is slightly faster than the baseline, but it is also rather clear that there is a lot of overhead, given the system CPU time. Additionally, this series of measurements also shows how independent steps are affected by the use of threading, even though this effect can not be properly qualified, since it does not seem to correlate with the overall run time of the legalization step.

When looking at the results, a threshold of 128 yields the lowest real time for a placement. However, at a threshold of 256 the real time used is only 30 ms higher, while requiring about a second less of user CPU time. As such, the threshold of 256 was chosen as the ideal value, and will be used during all following tests.

B. *Calculation of the gradients*

The next largest part of the duration of each iteration is the calculation of the gradients for each node. This calculation can be performed independently for each node, which means a simple OpenMP loop construct can be used. These allow

 ISBN: 978-1-61208-823-5

TABLE II. LIST OF THE TIMINGS FOR THE SEPARATE STEPS THAT OCCURRED FOR DIFFERENT SIZES OF THE WORK GROUPS FOR THE GRADIENT CALCULATION STEP

Work size	Legal. (μs)	Calc. Grad. (μs)	Apply Grad. (μs)	Calc. Nets (μs)	Time Real (s)	Time User (s)	Time Sys (s)
2	1492.2	374.3	73.8	241.6	26.84	87.49	1.350
4	1466.8	350.3	73.9	243.9	26.52	85.44	1.443
8	1465.0	336.1	75.8	245.9	26.39	84.15	1.392
16	1470.4	320.2	76.8	245.7	26.30	83.72	1.276
32	1484.6	318.7	77.6	244.7	26.38	83.99	1.462
64	1501.9	328.2	79.5	248.4	26.82	85.00	1.346
128	1501.1	335.6	80.2	252.3	26.96	84.81	1.553
256	1528.8	354.5	81.6	254.2	27.56	85.46	1.554
512	1524.0	378.6	81.3	252.9	27.77	86.18	1.333

TABLE III. LIST OF THE TIMINGS FOR THE SEPARATE STEPS THAT OCCURRED FOR DIFFERENT SIZES OF THE WORK GROUPS FOR THE NET INFORMATION CALCULATION STEP

Work size	Legal. (μs)	Calc. Grad. (μs)	Apply Grad. (μs)	Calc. Nets (μs)	Time Real (s)	Time User (s)	Time Sys (s)
2	1454.8	316.5	79.6	151.1	24.99	89.19	1.625
4	1459.4	315.5	80.1	126.6	24.83	86.98	1.970
8	1460.4	313.0	79.7	112.4	24.58	86.43	1.496
16	1458.1	316.6	80.3	105.8	24.50	85.94	1.737
32	1459.0	321.0	80.3	105.0	24.59	85.66	1.591
64	1460.4	325.2	80.5	107.2	24.67	85.45	1.810
128	1453.2	326.5	81.0	108.1	24.67	85.24	1.558
256	1458.9	322.5	80.3	109.2	24.62	85.67	1.665
512	1467.9	318.1	80.7	112.6	24.74	86.13	1.485

specifying the size of the work groups to be handled per thread, where the implications of the work group size are similar to the tasks for the recursive calls: Smaller work groups allow for higher parallelism, while larger work groups reduce the overall overhead. These work groups will be dispatched to available threads on a dynamic basis, meaning that if one thread happens to be scheduled more often by the OS, it may be able to accept two workloads in the same time as another thread accepts a single one.

The results for different work group sizes are shown in Table II. The row with threshold value 256 from Table I serves as the baseline in this case, since these results are supposed to improve on the ones for the legalization.

The difference between the worst and best times is much less pronounced here, compared to the results for the legalization step. At the same time, it is somewhat surprising that the entry with the lowest real time is with a work group size of just 16. This seems to counter the intuition that larger work groups should allow threads to work without interruption, and consequentially with lower overhead, but also mirrors the previous statement that there are various non-deterministic cross-thread effects.

The noted work group size of 16 was consequentially chosen as optimal and will be kept for successive tests.

C. Calculation of net information

The procedure used to evaluate the effect of multithreading on the net information calculation step is identical to the gradient calculation step. As before, these results attempt to improve upon the previous ones, so the row with a work group size of 16 from Table II now serves as the baseline. Table III shows the results for the net information calculation.

As can be seen the difference between the highest and the lowest real time measurement is even smaller than previously. However, the measurements are on average 2.1 seconds lower than the optimal duration for the gradient calculation step.

The lowest real time measurement for this step, if only by a small margin, is again with a work group size of 16.

D. Application of gradients to node position

Lastly, the application of the computed gradients was evaluated, again identically to the methodology applied during the last two steps. This step usually only takes a low percentage of the total time per iteration, so it is to be expected that the impact of using multiple threads will also be small.

The results are presented in Table IV. As before, the average real time duration is lower than the optimal run of the previous step, by about 1.1 seconds. In this instance, the lowest real time is for a work group size of 128. This can be explained by the fact that during the gradient application, each iteration of the loop performs very few memory accesses and much fewer operations in general, compared to the other steps. Here the duration would be dominated by the overhead for low work group sizes.

Interestingly, the user CPU time decreases by nearly 1.5 seconds, while in all other cases it increased when utilizing multiple threads. This might again be explained by the behavior of idle threads and general thread scheduling, since it is likely that there would be a certain amount of time wasted by the idle worker threads in OpenMP's thread pool, during the times, where the placement program uses only a single thread.

E. Summary

Overall we can observe a noteworthy improvement, where the real time requirement for a placement is reduced by 59.9 %, while increasing the required amount of user CPU time by about 45.9 %. At the same time the system CPU time increases by 5340 %, which is to be expected for heavy use of multithreading and thread synchronization.

 ISBN: 978-1-61208-823-5

TABLE IV. LIST OF THE TIMINGS FOR THE SEPARATE STEPS THAT OCCURRED FOR DIFFERENT SIZES OF THE WORK GROUPS FOR THE GRADIENT APPLICATION STEP

Work size	Legal. (μs)	Calc. Grad. (μs)	Apply Grad. (μs)	Calc. Nets (μs)	Time Real (s)	Time User (s)	Time Sys (s)
2	1406.2	308.0	113.0	97.7	24.14	89.65	1.780
4	1411.8	307.9	76.7	97.1	23.76	86.92	1.726
8	1417.9	306.2	55.0	97.6	23.56	85.25	2.039
16	1415.0	316.0	44.4	99.7	23.51	84.29	2.066
32	1418.2	319.8	40.8	100.6	23.61	84.44	1.801
64	1418.9	313.3	39.3	102.2	23.52	83.90	2.070
128	1414.1	304.7	39.1	99.3	23.24	84.45	1.632
256	1421.8	311.2	40.9	99.7	23.45	84.45	1.770

TABLE V. AVERAGE BOUNDING BOX COSTS AND STANDARD DEVIATION (IN PERCENT OF THE AVERAGE) OF THE RANDOM PLACEMENT FOR THE DIFFERENT NETLISTS

Net	Avg. BB	StdDev. (%)
s298	218.55	0.38
ex5p	174.03	0.41
apex4	191.11	0.43
alu4	202.73	0.44
misex3	201.05	0.59
tseng	100.16	0.64
elliptic	495.85	0.72
e64-4lut	30.57	0.80
seq	266.96	0.88
bigkey	202.97	0.99
diffeq	157.71	1.05
frisc	592.76	1.07
dsip	180.31	1.11
spla	664.16	1.26
apex2	290.97	1.34
s38584.1	767.97	1.35
s38417	801.45	1.37
pdc	965.86	1.48
ex1010	730.25	1.83
clma	1617.5	1.97
des	273.99	2.23

IV. IMPROVEMENT OF THE INITIAL PLACEMENT

As already explained, GPO just uses a random initial placement, which means that all CLBs and pins are placed at random, non-integer positions on the placement grid. For this, a simple PRNG with a constant seed is used, so the placement would stay identical between runs.

Of course the initial placement has a measurable effect on the end result, which will also affect later attempts to optimize the parameters. To get an idea of the magnitude of that effect, each netlist of the benchmark set was run through GPN multiple times, where the seed for the PRNG was changed each time to generate a different initial placement. Table V shows that the resulting bounding box cost values have a standard deviation of up to 2.23% of the mean, with different netlists being affected in different amounts. The most consistently placed netlist showed just 0.38% deviation. This poses a problem, when the different attempts at optimizing the algorithm are meant to get within the last percent of the bounding box cost that can be reached using Versatile Place and Route (VPR) [7].

This points out two facts:

1) The placement algorithm is, as is to be expected, not generally capable of finding the global minimum.
2) The overall quality of the placement could be improved by choosing a better initial placement.

As a first attempt a series of tests was run to see, if the bounding box cost of the initial random placement correlates with the cost of the end result. For this the nodes were placed randomly and a legalization pass was performed. After all nodes moved to their legal positions, the bounding box cost was calculated and noted. The exemplary results for the netlist "alu4" are shown in Figure 4 and turned out to have an R^2 of 0.00966 for a linear regression, which indicates no correlation. Thus, this measure can not be used to predict the quality of the final placement.

Next, the initial random placement was run through a low number of iterations (500) before the initial bounding box cost calculation, to see if this would give a more clear result, which is shown in Figure 5. As the plot shows, the results also do not correlate, with an R^2 of just 0.042 in this case, slightly better than the previous attempt, but still insignificant.

A. Starting placement via Min-Cut approach

Instead of initially placing all nodes at random, an approach inspired by the min-cut [8] problem was chosen. The placement area is split into two regions, and all nodes assigned to either one region or the other, such that neither region is overfilled. Then, for each node, the number of connections within the node's region is determined, as well as the number of connections to the other region. The two sums are then subtracted and assigned to the node as a "delta" value, i.e. an indication how many more connections would become internal, if the node would be moved to the other region. The following steps are then performed in a loop:

1) The lists of nodes and their delta values for both regions are sorted
2) One (and only one) of the following steps, selected in the given order:
   a) If region 2 is not full and the highest delta from region 1 is > 0 then move that node to region 2
   b) If region 1 is not full and the highest delta from region 2 is > 0 then move that node to region 1
   c) If the highest delta from region 1 is higher than the negation of the highest delta from region 2 then swap the corresponding nodes
3) The delta value of the moved node is updated, as well as for all nodes, which were connected to the node that was moved

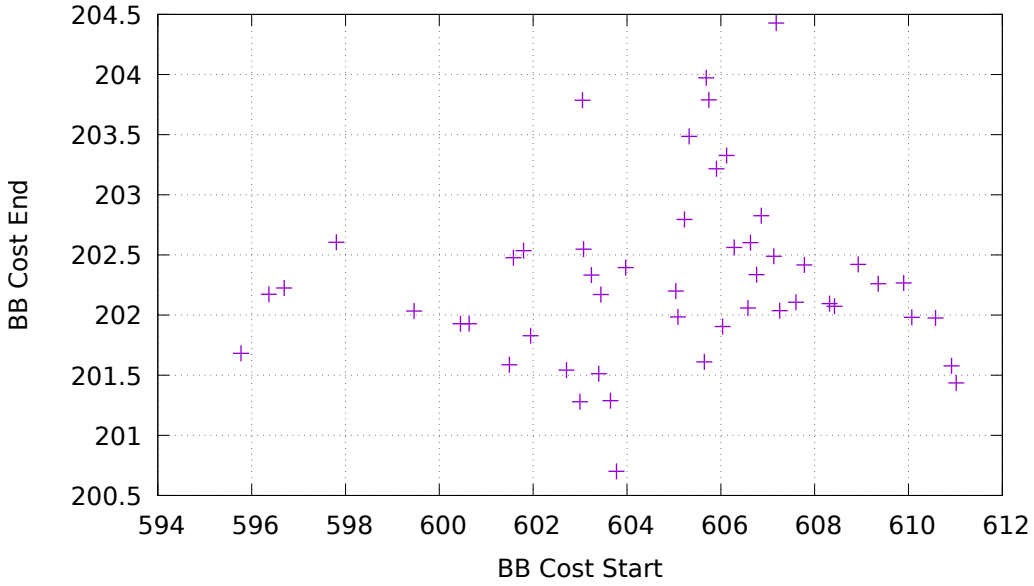


Figure 4. X/Y plot of the data points gathered to see if the BB cost of the initial random placement correlates with the end result

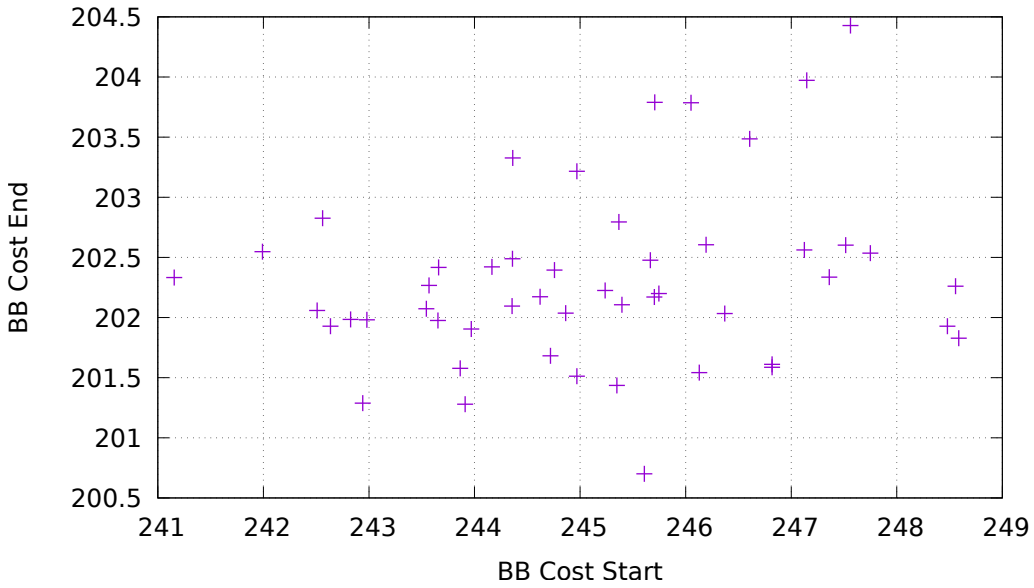


Figure 5. X/Y plot of the data points gathered to see if the BB cost after just 500 iterations correlates with the end result.

Step c) may switch nodes, even if one of the nodes has a negative delta, as long as the sum of the delta values is still positive. The loop is repeated until none of the steps a) through c) can be performed. In that case, a local minimum for the number of connections between regions was found.

Once this is the case, both areas are recursively split again and the same steps performed separately for the new regions. This is repeated until each remaining region ultimately only contain a single node, in which case that node will be placed at the resulting position. Since this recursion scheme functions much in the same way as the legalization step, the same multithreading improvement can be applied, even though the majority of the work happens in the initial region, which contains all nodes. Subdivided regions will contain a higher percentage of nodes, which have connections outside of both regions, and which will not be counted during the delta calculation.

The results for the nets, when placed after an initial min-cut run, are shown in Table VI, together with the relative percentage of the average bounding box cost for the random placement. As can be seen nearly all results are very close to the average, and most are even slightly below, with the “s298” net being the only outlier, having a final bounding box cost 9.64,% above the random average.

While not a major improvement, this means that the min-cut initial placement is slightly better than the random placement on average, and thus also avoids situations where the initial random placement may be detrimental to the quality of the final placement.

 ISBN: 978-1-61208-823-5

TABLE VI. BOUNDING BOX VALUES FOR ALL NETS AFTER PLACEMENT, WHEN THE DESCRIBED MIN-CUT APPROACH IS USED FOR INITIAL PLACEMENT, ALSO RELATIVE TO RANDOM INITIAL PLACEMENT

Net	Final BB	Rel. Rand. Avg. (%)
s298	216.3	109.64
ex5p	173.6	99.76
apex4	190.3	99.58
alu4	201.5	99.40
misex3	202.0	100.47
tseng	101.7	101.54
elliptic	503.5	101.54
e64-4lut	30.6	100.11
seq	266.2	99.72
bigkey	202.8	99.92
diffeq	156.3	99.11
frisc	566.9	95.64
dsip	177.8	98.61
spla	653.8	98.44
apex2	292.7	99.58
s38584.1	753.2	98.08
s38417	793.9	99.06
pdc	962.6	99.66
ex1010	725.3	99.32
clma	1592.5	98.45
des	276.8	101.02

V. IMPROVING THE TIMING OF THE PLACEMENT RESULTS

Another issue, which was rather common for GPO, was the path delay of the placed and routed netlists, which was on average about 46 % higher than results produced by VPR. This is due to the fact that VPR uses not only the previously presented cost metric for the bounding box, but also uses a metric for the timing behavior, to keep the critical path as short as possible, which GPO did not.

To remedy this shortcoming a kind of path metric had to be calculated. This was implemented by finding the maximum number of nodes preceding and following each node, where an input has zero preceding nodes, and an output zero following nodes respectively. Clocked logic elements act as inputs/outputs respectively, since in terms of timing analysis they break the critical path. These two measures are then summed, if the node is not clocked, or the maximum of both is taken if it is, to yield the path metric of that node, meaning that all nodes directly on the critical path also have the highest path metric. After all nodes have been processed the path metric of all nodes is brought into the range $[0; 1]$ by the simple formula

$$p'_b = e^{\alpha_3 \cdot (p_b - p_M)}$$

where p_b is the previously calculated path metric, and p_M is the highest path metric found. Consequentially, nodes that are on the critical path have a path metric of 1, whereas nodes which are on an even slightly shorter path will have a very low path metric. For example, if the path a node is on is four nodes shorter than the critical path, its path metric will only be 0.018 when $\alpha_3 = 1$, which means the node will have very little impact on later calculations.

With the path metric depending only on the node difference between longest and current path the drop-off is always similar between netlists with different amounts of nodes and thus different lengths of the critical path. A variant, where the path metric uses the percentage of nodes on the path, relative to the longest one, i.e. $\frac{p_b}{p_M}$, would cause the drop-off to be much smaller in netlists that have a much longer critical path. For example in a netlist with a critical path of length 20, a node on a path of length 19 would have a path metric of 0.369 with the chosen function, whereas the ratio would still be 95 %, giving too much importance to nodes not on the critical path. Raising that ratio to some power would alleviate the effect, but the power would have to depend on the length of the critical path, otherwise the problem would just shift slightly.

The parameter α_3 allows to change how quickly the path metric decays. The higher the parameter is the fewer nodes will be strongly affected by the path metric aspect, and vice versa. However, if there are too many nodes being affected by the path metric step, it stops being useful to stabilize the critical path.

This path metric is then used during the gradient calculation. For every node, a few terms are calculated, based on the nodes following and preceding it. The following formulas describe the calculations for the nodes preceding the current node, which are basically identical to the formulas for the nodes following the current one.

$$g_{x_p}(n) = \sum_{n_2 \in N_{prev}} (x_n - x_{n_2}) \cdot p_{n_2} \tag{4}$$

$$g_{y_p}(n) = \sum_{n_2 \in N_{prev}} (y_n - y_{n_2}) \cdot p_{n_2} \tag{5}$$

$$w_p(n) = \sum_{n_2 \in N_{prev}} p_{n_2} \tag{6}$$

$$g'_{x_p}(n) = g_{x_p} \cdot \frac{\alpha_4 \cdot p_n}{w_p(n)} \tag{7}$$

$$g'_{y_p}(n) = g_{y_p} \cdot \frac{\alpha_4 \cdot p_n}{w_p(n)} \tag{8}$$

N_{prev} is the set of all preceding nodes, which are connected to the given node. p_n refers to the path metric of node n, as previously calculated, so that $g_{x_p}(n)$ and $g_{y_p}(n)$ are the weighted sums of position differences to all preceding nodes. $w_p(n)$ is the sum of the path metrics of all preceding nodes, and is used in $g'_{x_p}(n)$ and $g'_{y_p}(n)$ to normalize the gradient terms, so that the terms between a node with very few preceding or following nodes, and one with many are similar in magnitude. $g'_{x_p}(n)$ and $g'_{y_p}(n)$ are then the final terms, which are added to the gradients for the given node. The new parameter a_4 allows to control, how strong the effect of the path metric terms on the node's gradients should be.

After a few manual tests, a full test run was performed using $\alpha_3 = 0.75$ and $\alpha_4 = 0.2$, the results of which are shown in Table VII. The averages indicate that the bounding box cost of the placement is overall still slightly worse than VPR, if only by 1.36 %, but also improves on the results of GPO by 0.52 %. The timing of the critical path is still noticeably worse than with VPR, but improves on the previous results by 11.86 %.

Since the manipulation of gradients for the path length aspect affects the general placement flow, optimization of the parameter α_4 by itself is not sensible. Thus, the final choice of the parameter α_4 will be decided during the general attempt to optimize the various parameters of the algorithm.

VI. OPTIMIZATION OF PARAMETERS

Since, so far, the various parameters of the algorithm have only been chosen manually after a few empirical tests, it is

 ISBN: 978-1-61208-823-5

TABLE VII. RESULTS FOR THE VARIOUS NETLISTS OF THE FIRST ATTEMPT OF UTILIZING PATH LENGTH DURING OPTIMIZATION, RELATIVE TO VPR AND GPO

Netlist	BB. Rel. (%)		Crit. Path. Rel. (%)	
	VPR	GPO	VPR	GPO
e64-4lut	100.56	97.43	104.39	87.34
tseng	99.68	101.84	171.67	106.46
ex5p	96.15	99.09	148.32	85.57
apex4	98.17	100.90	101.13	96.61
dsip	89.39	98.20	102.93	73.20
misex3	101.14	101.59	121.83	78.60
diffeq	101.33	101.66	178.29	103.86
alu4	99.82	101.52	149.74	110.81
des	106.26	103.98	150.86	103.61
bigkey	97.85	101.92	116.20	87.03
seq	102.36	99.67	93.45	76.54
apex2	102.69	98.88	112.70	84.07
s298	95.69	98.62	141.27	81.02
frisc	95.79	93.67	154.16	78.95
elliptic	100.10	99.33	178.81	99.91
spla	105.06	97.93	107.58	83.53
pdc	103.28	101.07	88.25	83.98
ex1010	105.24	99.08	104.91	87.55
s38417	114.62	101.31	143.65	79.26
s38584.1	109.82	95.80	135.73	90.71
clma	103.51	95.65	94.43	72.25
Average	101.36	99.48	128.57	88.14

very likely that better parameter values could be found by performing an actual parameter search. The parameters in question are specifically:

1) The global step size of the Adam optimizer
2) The legalization factors for pins and CLBs
3) The factors α_1 and α_2 of the previously described gradient function
4) The Factor α_4 of the path length aspect

Since there are multiple phases, different sets of parameters are required depending on the progress of the placement algorithm. With two sets consisting of six parameters each, every phase has twelve parameters in total.

On one hand, blindly picking parameters and trying them on the different netlists would not be helpful in determining the optimum. On the other hand, an exhaustive search of the parameter space would be infeasible, since every invocation of GPN takes several seconds at best, and exhaustively searching a 12-dimensional parameter space would require an incredibly high amount of invocations.

The problem can be generalized to finding the (ideally global) minimum of a not fully known cost function with 12 variables, which is expensive to evaluate, using as few evaluations of the function as possible. Usually, this could be simplified by using an approximation of the cost function and searching for a minimum on this approximation. For functions of very few variables it is usually feasible to use a regression to a quadratic or cubic function, but with the given 12 variables this approach becomes rather problematic. Instead, the search space was investigated using artificial neural networks (i.e., the purely mathematical layered model, as commonly utilized in machine learning).

In a general sense, a neural network can approximate any function of a certain amount of input variables, where the accuracy of the approximation is limited by the complexity of the network, as well as the number of known input/output pairs used during training. The exact complexity of the used network is usually a compromise between the precision of the results and an attempt to keep the network from overfitting. Overfitting in this context means that the network gives the exact results for the known data pairs, but is completely wrong on any other point. This occurs when the network has too high complexity for the dataset, so instead of generalizing, the network only detects, which of the known inputs was given, and produces the specific output value for that input.

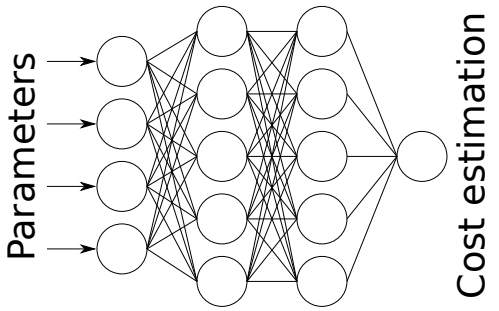


Figure 6. Schematic visualization of the used network layout.

The network layout used is a simple feed-forward type network, schematically shown in Figure 6, utilizing an input layer with as many nodes as parameters, two hidden layers with 40 nodes each, and an output layer with a single node, representing the estimated cost value. The used activation function is the exponential linear unit (ELU), and the training uses the Adam optimizer. Each netlist has its own instance of such a network, since the relation between parameters to cost is different for each netlist.

A. *Preparation of input and output values*

Artificial neural networks tend to work better when the inputs are normalized to the same range. This means that the different parameters, which each have different ranges, are all mapped to the range $[0; 1]$, given a minimum and maximum for each parameter. This also ensures that the chosen value for a certain parameter never goes outside of the sensible range, for example the step size of the Adam optimizer should never be negative, and ideally also never be zero, since a value of zero means no movement of nodes. Similar restrictions apply for the other parameters. The chosen ranges for the parameters are thus:

- **Adam step size:** $[0.01; 1.0]$
- **Legalization factors:** $[0.0001; 1.0]$
- **Net-size cost factors α_1 and α_2:** $[0.05; 20.0]$
- **Path length factor α_4:** $[0.01; 1.0]$

The output also needs to be normalized from the different bounding box costs of the netlists to the range $[0; 1]$. This is achieved the same way as the parameters, except the minimum and maximum values are dynamic, based on the lowest and highest cost that has been found during actual evaluation of the different sets of parameters. However, the range of cost values will generally be quite large, where the worst result might be as much as three times as high as the best one. At the same time, the majority of results are in the lower cost range, which means that the output range would be rather squashed towards zero. Since we are only really interested in the low range the

normalized output value should be slightly skewed, such that the low cost range spans a larger part of the $[0; 1]$ interval. For this, the square root of the normalized output values will be used as the training target, which is still in the given range, but stretches low values over a wider part of the interval.

B. Training process

The training process initially operates separately on the different netlists. It starts by trying 32 random sets of parameters for each netlist and noting the results. These are then used to start the training of the corresponding network, until the network's loss drops below a threshold of 0.01, where the loss is calculated as the sum of squared errors between the target and calculated output values.

The network is then used to determine a set of eight new parameter sets. This is done by utilizing the normal gradient descent approach, which is also used during training, but instead of changing the network's weights, the input is changed to minimize the output. For this, a constant negative gradient is fed backwards through the network, and the calculated gradients for the inputs are then used to slightly change these. This is repeated 2000 times, and the resulting parameter sets, after they have been brought back to the proper ranges, again evaluated using GPN. The (normalized) parameters are clamped to the range $[0; 1]$ while iterating, to ensure they do not leave the predefined sensible range.

The whole process is repeated a number of times, where first improved results may occur within 50 known points, and more proper results usually occur within the first 200 known points. After 200 iterations, the approach is switched from a per-netlist to a shared one.

The general procedure stays the same in the shared approach, with the difference that the next parameter sets to evaluate is determined by using all the trained networks. So instead of finding a per-netlist cost minimum, a minimum for all used netlists is determined by summing their outputs. The outputs are not brought back into their normal range, so each net has the same weight on the cost sum, whereas the weight would otherwise depend on the range of the bounding box costs. The parameter set is then evaluated on the given netlists, and all networks re-trained.

C. Intermediate results

For the optimization, GPN has been slightly simplified to use only a single placement phase, instead of multiple as before, and also perform only 6,000 iterations, to speed up initial attempts. It would also load the parameters from a file, instead of using hardcoded default values, which was required to allow parameter sets to be specified by the training process. During the single optimization phase, all parameters are linearly swept from the "before" to the "after" values, i.e. all values slowly change during placement.

Additionally, the number of netlists for the search is only a subset of the full benchmark, to reduce the required time per iteration to a more reasonable amount.

During the initial attempts, it already became clear that even at 6,000 iterations the results were either pretty close to the ones produced by VPR, or sometimes even better. Additionally, since the parameters were optimized separately for each netlist in the first step, it is possible to find a separate parameter set for nearly every netlist, which leads to a better placement (in terms of the bounding box costs) than VPR.

TABLE VIII. INTERMEDIATE RESULTS FOR THE SEPARATE NETLISTS DURING PARAMETER OPTIMIZATION

Netlist	BB. Cost	Rel. VPR (%)	Rel. GPO (%)
e64-4lut	29.7	96.9	93.9
tseng	96.7	94.4	96.5
ex5p	169.3	93.7	96.6
apex4	188.7	96.6	99.3
dsip	176.2	88.2	96.9
alu4	200.3	97.9	99.5
des	252.5	98.0	95.9
apex2	286.4	102.2	98.5
frisc	555.1	94.5	92.5
elliptic	483.0	97.0	96.3

TABLE IX. FINAL CHOICE OF PARAMETERS

Parameter	Value Pre.	Value Post.
Adam Step size	0.934	0.346
CLB Legalization	0.0438	0.460
Pin Legalization	0.0001	0.506
Gradient factor α_1	0.050	14.978
Gradient factor α_2	13.267	12.267
Path length factor α_4	0.591	0.219

Table VIII shows the intermediate results, when using the best found parameter set for every single netlist. It improves on the results from GPO for all tested netlists, and in nearly all cases when compared to VPR. This indicates that the initial approach, utilizing multiple phases and 12,000 iterations, was in fact not necessary to achieve good results, so the simplified placement algorithm with reduced iteration count will also be sufficient.

D. Chosen parameters

The shared training process was continued until no improvements could be observed for 100 iterations. Then, the best found parameter set was chosen as the final parameters that GPN will use, which are shown in Table IX.

It can be seen that the step size of the Adam optimizer reduces during the placement phase, which mirrors prior observations that a higher step size is usually useful for rough organization, while a low step size is required for proper fine detail placement towards the end. This is similar to the slowly declining "temperature" of the simulated annealing algorithms like the one VPR uses.

The legalization factors for CLBs and pins increase during placement, starting with a rather low value, to allow pins and CLBs to quickly move to a more ideal region at the beginning.

For the gradient calculation, the factor α_2 turns out to change only slightly, whereas factor α_1 increases from close to zero to about 75 % of the allowed range, indicating contraction of the bounding boxes for the netlists to be more important towards the end of the placement process. Inversely, the factor α_4 for the path length metric decreases during placement, which leads to the critical path being prioritized especially in the beginning.

The results of applying the chosen parameters to all netlists in the benchmark will be shown in Section VII.

VII. FINAL RESULTS

The final version of GPN, after the various attempts to optimize performance, was again benchmarked using the

 ISBN: 978-1-61208-823-5

TABLE X. A LIST OF THE USED BENCHMARKS AND THEIR CHARACTERISTICS, THE NUMBER OF CLBS, INPUT BLOCKS, OUTPUT BLOCKS AND THE SUM OF ALL NODES

Name	Inputs	Outputs	CLBs	Nodes
ex5p	8	63	1064	1135
tseng	52	122	1047	1221
apex4	9	19	1262	1290
misex3	14	14	1397	1425
alu4	14	8	1522	1544
diffeq	64	39	1497	1600
dsip	229	197	1370	1796
seq	41	35	1750	1826
apex2	38	3	1878	1919
s298	4	6	1931	1941
des	256	245	1591	2092
bigkey	229	197	1707	2133
frisc	20	116	3556	3692
spla	16	46	3690	3752
elliptic	131	114	3604	3849
ex1010	10	10	4598	4618
pdc	16	40	4575	4631
s38417	29	106	6406	6541
s38584.1	38	304	6447	6789
clma	62	82	8383	8527

TABLE XI. COMPARISON OF THE BOUNDING-BOX COSTS BETWEEN THE GRADIENT PLACEMENT AND THE SIMULATED ANNEALING OF VPR

Netlist	VPR	GPO	GPN (Pt. / % VPR / % GPO)		
ex5p	180.599	175.250	170.872	94.61	97.50
tseng	102.398	100.219	100.147	97.80	99.93
apex4	195.338	190.064	190.817	97.69	100.40
misex3	200.456	199.570	202.401	100.97	101.42
alu4	204.692	201.253	203.429	99.38	101.08
diffeq	155.531	155.028	157.706	101.40	101.73
dsip	199.845	181.925	180.203	90.17	99.05
seq	260.789	267.835	264.410	101.39	98.72
apex2	280.120	290.910	289.174	103.23	99.40
s298	225.344	218.734	211.962	94.02	96.90
des	257.643	263.300	262.286	101.80	99.61
bigkey	209.470	201.106	202.395	96.62	100.64
frisc	587.227	600.463	556.471	94.76	92.67
spla	628.155	673.901	679.609	108.19	100.85
elliptic	497.645	501.466	510.023	102.49	101.71
ex1010	684.798	727.315	733.547	107.12	100.86
pdc	939.813	960.346	959.026	102.04	99.86
s38417	687.198	777.488	733.140	106.69	94.30
s38584.1	684.220	784.347	745.078	108.89	94.99
clma	1502.330	1625.850	1569.470	104.47	96.53
Average				100.57	98.73

MCNC set of netlists, which were also used in the original paper, and some of which were used to evaluate parameters for the intermediate stages. The results shown are compared to VPR, but this time also with GPO, to give an indication how significant the effect of the optimization actually is.

The set of netlists used for benchmarking is listed in Table X, including their complexity as the number of nodes. The netlists are technology-mapped for, and will be placed on a simple island-style FPGA-architecture with four bit lookup tables.

A. Bounding-Box Costs

As before, the main cost metric employed by VPR is the simple sum of all net bounding boxes, as given by the formula

$$Cost = \sum_{n=1}^{N_{nets}} q(n) \left[\frac{bb_x(n)}{C_{av,x}(n)} + \frac{bb_y(n)}{C_{av,y}(n)} \right] \quad (9)$$

where $bb_x(n)$ and $bb_y(n)$ describe the size of the bounding box for net n in x and y direction. $q(n)$ is a corrective factor, which compensates for the expected routing effort. What this means is that a net with just two nodes would need just as many routing resources as required to span the distance of the bounding box, whereas a net with more nodes would need additional routing segments inside the bounding box to properly connect all nodes. However, the number of routing segments required is not linear to the number of nodes, since many nodes will be either close to each other, or able to share routing segments with nearby nodes of the same net. In short: While a net with more nodes will always require more resources, the relative routing overhead is much lower for nets with many nodes than for nets with very few nodes, so multiplying the size of the bounding box by the number of nodes would noticeably overestimate the required routing effort.

This $q(n)$ term is elaborated in two parts: For numbers below 50, a simple table is used, whereas for numbers ≥ 50 a linear function is utilized. The mentioned table holds a factor of 1.0 for up to three nodes, and a logarithmically increasing sequence for larger values. This sequence can be approximated with the following logarithmic function:

$$q'(n) \approx 1.0 + 8.543 \cdot \ln\left(0.953 + 0.0234 \cdot n^{0.635}\right) \quad (10)$$

n in this case refers to the number of nodes in a given net. The linear function is then just a continuation, given as

$$q''(n) \approx 2.7933 + 0.02616 \cdot (n - 50) \quad (11)$$

Table XI shows the resulting bounding box costs for the three algorithms after a complete placement run, and their relative percentage to each other. As can be seen, GPN is on average still slightly worse than VPR (by 0.57,%), but improved on the results of the original version by 1.27 %.

B. Runtime

The runtime behavior of the final version of GPN was also reevaluated. Given that one of the early optimization was utilization of multithreading, which by itself already showed a good deal of improvement, a noticeable reduction in the runtime of the placement would be expected.

Table XII and Figure 7 show the runtime results, again comparing the final version of GPN to VPR and GPO. As can be seen, the algorithm is now a bit more than five times faster than VPR, while also being 2.16 times faster than GPO.

A noteworthy detail is how the runtime changes in respect to the number of nets in each netlist. A regression of this relationship to a function of the form $\alpha_1 \cdot x^{\alpha_2}$ was performed, with the results shown in Figure 8. It can be seen that VPR's runtime increases faster with the number of nets than the one of the presented algorithm. While the factor α_2 is about 1.173 for GPO and GPN, it is approximately 1.46 for VPR, meaning that the difference in placement time will only get more pronounced the more nets a netlist contains.

 ISBN: 978-1-61208-823-5

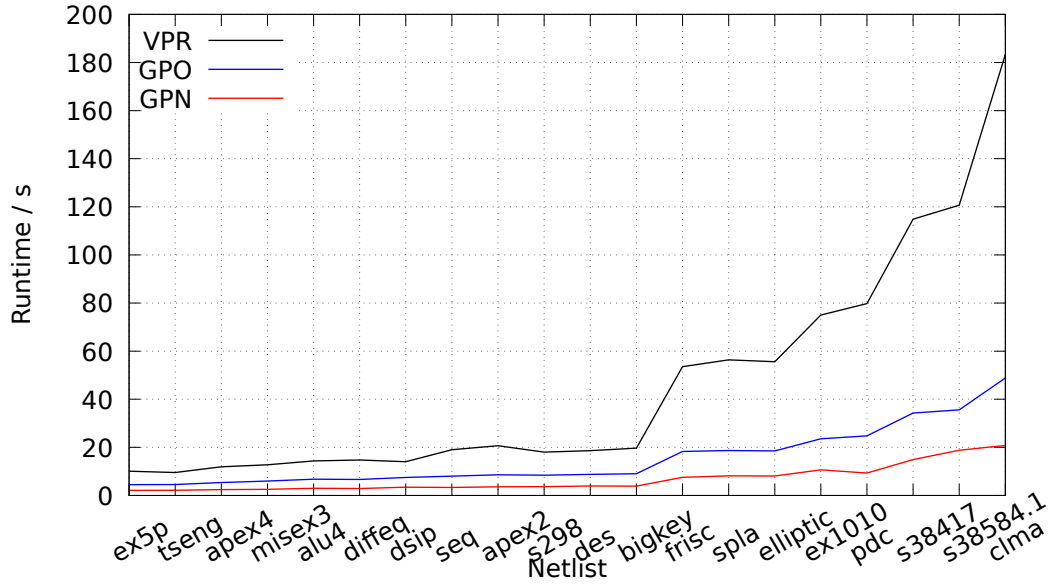


Figure 7. Diagram of the runtime as average of ten measurements between the gradient based placement algorithm (previous and improved) and the simulated annealing of VPR.

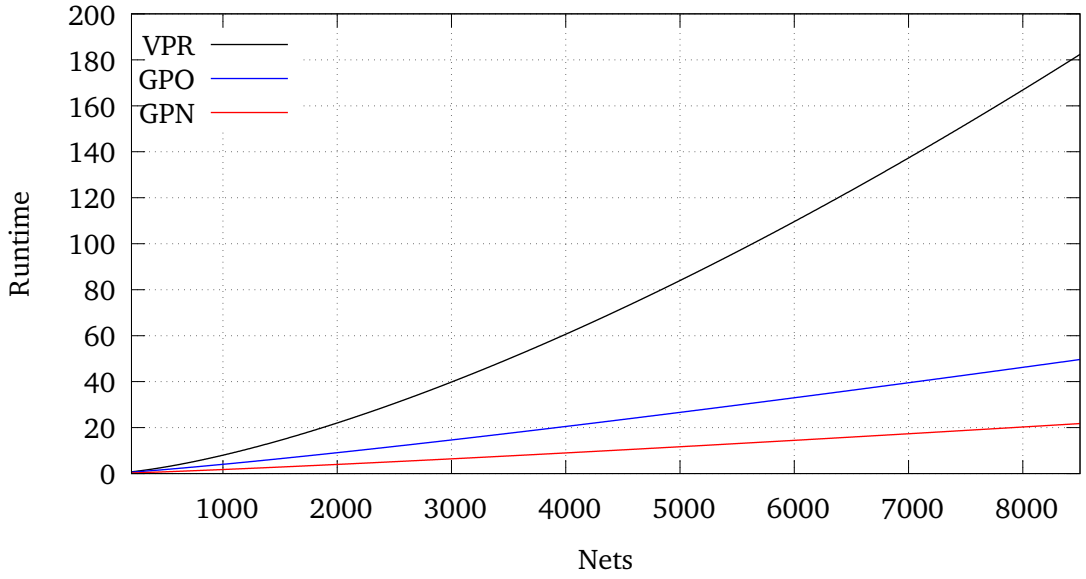


Figure 8. Plot of the regression of run time over the number of nets per netlist.

C. *Timing*

The critical path lengths were not reported in the original paper, but they were significantly worse compared to VPR. One of the optimizations, namely the addition of the path metric mechanism, explicitly addreses this issue. The results are shown in Table XIII.

While GPO was worse for every single netlist, the final version of GPN at least delivers better results than VPR for three of the netlists. In general, the results have improved by 16 % over the original version, but are on average still 21.14 % worse compared to VPR.

VIII. CONCLUSION

This paper shows that applying multiple threads to several steps of the placement algorithm speeds up the overall placement, such that GPN is on average 5.1 times as fast as VPR. It also shows that the run time of GPN increases slower with the number of nets in the netlist compared to VPR, which indicates that GPN scales better. At the same time simply using more threads does not always yield an improvement,

 ISBN: 978-1-61208-823-5

TABLE XII. COMPARISON OF THE PROGRAM RUN TIME BETWEEN THE GRADIENT BASED PLACEMENT ALGORITHM (PREVIOUS AND IMPROVED) AND VPR

Netlist	VPR (s)	GPO (s)	GPN (s / % VPR / % GPO)		
ex5p	10.08	4.48	2.10	20.86	46.93
tseng	9.54	4.57	2.19	22.91	47.83
apex4	11.90	5.36	2.43	20.39	45.32
misex3	12.77	6.00	2.56	20.02	42.60
alu4	14.33	6.74	2.99	20.89	44.45
diffeq	14.73	6.67	2.91	19.75	43.60
dsip	14.03	7.51	3.46	24.64	45.99
seq	18.99	8.02	3.32	17.49	41.39
apex2	20.67	8.61	3.62	17.54	42.11
s298	18.02	8.41	3.66	20.33	43.55
des	18.63	8.75	3.96	21.26	45.27
bigkey	19.69	9.05	3.90	19.83	43.16
frisc	53.52	18.31	7.54	14.08	41.17
spla	56.36	18.69	8.16	14.48	43.66
elliptic	55.61	18.50	8.09	14.54	43.72
ex1010	75.01	23.57	10.62	14.16	45.05
pdc	79.45	24.79	9.30	11.66	37.52
s38417	114.86	34.24	14.89	12.96	43.49
s38584.1	120.65	35.57	18.77	15.55	52.75
clma	183.35	48.81	20.81	11.35	42.63
Average				19.69	46.39

TABLE XIII. COMPARISON OF THE CRITICAL PATH DELAY BETWEEN THE GRADIENT BASED PLACEMENT ALGORITHM (PREVIOUS AND IMPROVED) AND VPR

Netlist	VPR (ns)	GPO (ns)	GPN (ns / % VPR / % GPO)		
ex5p	77.43	134.21	97.76	126.25	72.84
tseng	54.05	87.16	80.22	148.42	92.04
apex4	115.96	121.39	101.22	87.29	83.39
misex3	80.33	124.52	94.37	117.13	75.79
alu4	81.17	109.50	119.98	147.81	109.58
diffeq	64.47	110.67	106.33	164.93	96.08
dsip	62.06	87.26	72.69	117.13	83.30
seq	112.07	136.84	136.83	122.09	99.99
apex2	96.96	129.98	126.04	129.99	96.97
s298	146.11	254.77	173.73	118.90	68.19
des	89.68	130.58	116.57	129.98	89.27
bigkey	62.56	83.53	72.67	116.16	87.00
frisc	133.22	260.13	177.07	132.92	68.07
spla	158.94	204.70	182.83	115.03	89.32
elliptic	138.25	247.43	196.62	142.22	79.46
ex1010	181.87	217.93	198.90	109.37	91.27
pdc	232.23	244.01	197.77	85.16	81.05
s38417	123.94	224.62	135.77	109.55	60.45
s38584.1	94.18	140.92	136.63	145.07	96.96
clma	231.10	302.03	197.55	85.48	65.41
Average				121.14	84.00

next to the fact that there is a natural limit on how much faster the overall placement can get, depending on the amount of inherently sequential components.

It was also evaluated, how much the initial placement affects the final results, and a better approach to generate the initial placement was tested and implemented. This makes sure that the initial placement is always about as good as random placements would be on average. At the same time initial random placements which severely reduce the quality of the final result are avoided.

The timing behavior of the critical path, which the previous paper did not fully address, was also improved by about 16 % on average. This was achieved by introducing another gradient term for each node, which depends on the node's path metric, and causes nodes on the critical path to be placed closer to each other.

The various parameters were optimized using a simple feed-forward type neuronal network, which is used as a dynamic approximation of the cost function, since direct evaluation of the cost function via program invocations takes a prohibitively long time. During the parameter optimization, it was also discovered that the approach of using multiple placement phases was not in fact necessary. As a result, the final number of iterations was halved. It was also shown that, unsurprisingly, the best results for each netlist could be achieved by using a parameter set specifically optimized for that netlist, instead of using a single parameter set for all netlists.

Finally, it has to be acknowledged that the version of VPR used in this work as benchmarking platform is quite outdated. In future work, a recent version of the Verilog-To-Routing (VTR) Project for FPGAs [9] should be used instead. Even though the gradient placement approach was shown to be comparably fast for large netlists, a more recent set of benchmarks like the one included in VTR – containing much larger netlists – could be used to underline the scalability of the approach even further.

REFERENCES

[1] T. Bostelmann, T. Thiemann, and S. Sawitzki, "Fast FPGA-placement using a gradient descent based algorithm," International Journal on Advances in Systems and Measurements, vol. 13, no. 1 & 2, 2020, pp. 175–184.

[2] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," Microelectronics Center of North Carolina, Tech. Rep., 1991.

[3] G. Marsaglia, "Xorshift RNGs," Journal of Statistical Software, Articles, vol. 8, no. 14, 2003, pp. 1–6.

[4] M. Gort and J. H. Anderson, "Analytical placement for heterogeneous FPGAs," in 22nd International Conference on Field Programmable Logic and Applications (FPL), August 2012, pp. 143–150.

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings, 2015, pp. 1–15.

[6] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," IEEE Computational Science and Engineering, vol. 5, no. 1, 1998, pp. 46–55.

[7] V. Betz and J. Rose, "VPR: A new packing, placement and routing tool for FPGA research," in International Conference on Field Programmable Logic and Applications (FPL). Springer, 1997, pp. 213–222.

[8] M. Stoer and F. Wagner, "A simple min-cut algorithm," Journal of the ACM, vol. 44, no. 4, Jul. 1997, pp. 585–591.

[9] K. E. Murray et al., "VTR 8: High performance cad and customizable FPGA architecture modelling," ACM Trans. Reconfigurable Technol. Syst., 2020.

 ISBN: 978-1-61208-823-5